ANSI E1.17-2006 Architecture for Control Networks – Device Description Language (DDL)

This document forms part of ANSI E1.17-2006, Entertainment Technology - Architecture for Control Networks, which was approved as an American National Standard by the ANSI Board of Standards Review on 2006 October 19

ESTA TSP document ref. CP/2003-1011R4

	Revision History
Revision R4	2006-03-14
Revision R4pre2	2006-03-08
Revision R4pre1	2006-03-03
Revision R3	2005-10-14
Revision R3pre2	2005-05-20
Revision R3pre1	2005-04-25
Revision R2	2004-11-09
Revision R1	2004-10-07
Revision Pre1	2003

Abstract

DDL is a language for describing devices controllable by getting and setting of properties (this can apply to almost anything!). This document 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] which 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:

The device model – an abstract model used for describing devices.

The XML format for DDL – a syntax for rendering device descriptions in a machine and human readable serial format.

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 which 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 which 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 which identifies a device class is called a *Device Class Identifier* or *DCID*.

3.3. Modules, Devices and Appliances

DDL *devices* are modules which 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 which 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* which exposes devices see Section 2.1, “Relationship Between DDL Devices and DMP Components”)

While a DDL device is a module, DDL syntax also defines other modules which are not devices. See Section 4.5, “DDL Modules – Devices, Behaviorsets and Languagesets”.

3.4. Trees of Properties

The hierarchy of properties representing a device in DDL forms a set of modified tree structures.

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* which has no parent itself but is the parent, grandparent 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*.

In DDL each node of the tree is called a property and one or more trees of properties represents the structure of a device. As well as zero or more child properties (also called sub properties or properties of properties) each property shall have zero or one values.

The strict tree structure is modified by a limited ability to add cross-links between arbitrary properties so that the same property can be the child of more than one parent node.

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 document. 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 that 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

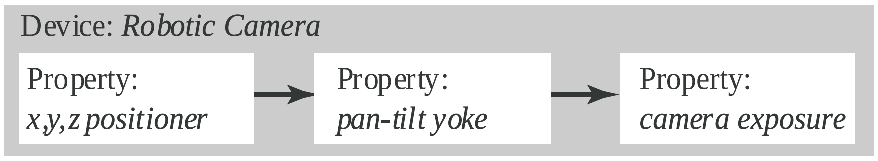
Device: *Robotic Camera*
Property: *x,y,z positioner*
Property: *pan-tilt yoke*
Property: *camera exposure*

Figure 2. 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 which 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 3. Robotic Camera Extended Property Layout

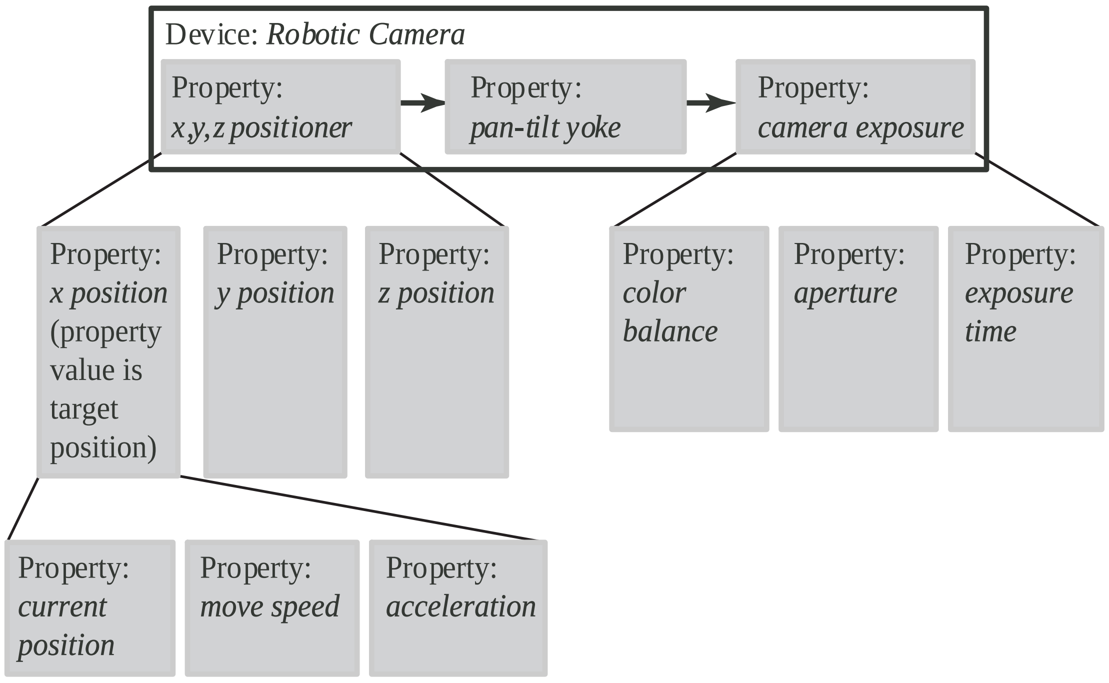
Device: *Robotic Camera*
Property: *x,y,z positioner*
Property: *pan-tilt yoke*
Property: *camera exposure*
Property: *x position* (property value is target position)
Property: *y position*
Property: *z position*
Property: *color balance*
Property: *aperture*
Property: *exposure time*
Property: *current position*
Property: *move speed*
Property: *acceleration*

Figure 4. 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 5, "Simple Dimmer Pack Property Layout" 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 5. Simple Dimmer Pack Property Layout

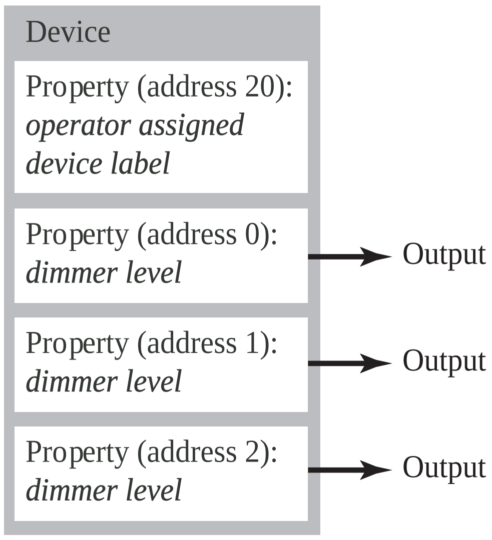
Device
Property (address 20): *operator assigned device label*
Property (address 0): *dimmer level*
Output
Property (address 1): *dimmer level*
Output
Property (address 2): *dimmer level*
Output

Figure 6. 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>
```

3.4.1. Meaning of Property Values – Driven Properties

The value of a property should always represent the actual state of the device. In many cases there is a difference between a property representing a physical position or state and one representing a desired or target state.

Take for example, a property representing the *desired* position of a winch. When it is set to a new value, the device must ramp up the motor speed, move for some time at a given speed and finally 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. Furthermore the property representing the actual position could not be writable.

In this example, the actual position property is driven by the desired position property and by the algorithms governing how the winch moves in response to changes in target position. These algorithms are often governed by a wide range of parameters such as maximum speed, acceleration and so on each of which is a property in its own right. The target position, speed limit, acceleration etc. are said to be drivers of the winch position.

The roles of “driven”, “target” and other driver properties are expressed by their behavior (see below). In DDL driver properties shall always be children or descendants of the properties they drive. The parent-child relationship, together with the many refinements of driver and driven behavior indicates to the controller, how the writable driver properties are combined to produce the required driven output.

In the winch example the actual position property cannot be written directly over the network as this would imply instantaneous movement. In general most properties representing physical features are likely to need to be driven via a target value and probably other driver properties. However, with some features, a direct write is quite possible, either because the property represents something which does not have direct physical representation or because the effect is effectively instantaneous – e.g. setting a preamplifier gain.

Another example of a “driven” property would be a lighting dimmer which has multiple inputs each of equal weight, which are combined on a “highest takes precedence” basis as is common in lighting control. In this case, there is not just one target property but several all of which have equal weight and the primary “driven” property reflects the combination of all the targets according to the highest value algorithm.

In the case of a linear “slider” input, the primary property would look very similar to the winch - it represents the physical feature of the device (a linear displacement between limits). However, because the slider is not driven by any writable property it must be treated as an input.

3.4.1.1. Implied Properties

In some cases the value of a primary driven property may not be available – for example, because it is an analog servo driven system and the controller may have no feedback of actual position, or simply because the data link is unidirectional. In these cases the property is neither writable nor readable – this is called an implied property. Implied properties do not require network addresses and their values are unavailable or must be inferred from the properties driving them. They must however be declared as this is how DDL builds a consistent and rigorous description of the device. Implementers should avoid implied properties wherever a readable value could be provided.

3.4.1.2. Unattainable targets

If a target state is set (by writing a value to a property) which is a legal operation within the protocol, yet which breaks constraints on that property (e.g. a value which exceeds a limit, or which 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 which is close to that requested. DDL imposes no requirement on which choice is taken. However, *access protocols* may define their own rules or may provide messages to handle this situation. Property behaviors may also be defined which allow control or expression of which option is taken or which show the attainable target which has been adopted.

3.4.2. Shared Properties

A straightforward tree (or nested container) structure is not adequate to describe all devices likely to be encountered. 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 – a property which appears in multiple places in a description shall be declared to be shared.

Syntactically, this is achieved in XML by declaring an identifier for the property. All properties which are declared to be shared and which have the same identifier are in fact the same property appearing in multiple places.

It is an error if the attributes or content of the same shared property differ in meaning in different declarations.

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 Section 3.3, “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 which allow the sub device type to change or be re-specified are very powerful in describing equipment which 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 which 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 Section 7, “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 which are distinguished by where and how their value, if any, is located.

NULL
: We have seen (Section 3.4, “Trees 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 (Section 3.6, “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. (Section 3.4.1.1, “Implied Properties”)

Immediate
: The fourth kind of property value is an immediate value. This is a value which 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 which 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). 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 which 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:

Figure 7. 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*. Property *behaviors* are extensible and each is derived by refinement from previously defined behaviors (a little like “typedef” in C). Thus, the *maximum* type in the example could be derived from the *limit* type. Other types could then be refined from *maximum* if required. The meaning of *maximum* is something that a controller might need to have code associated with in order to use the information or simply to decide how to present it to the user.

For a rich set of behavior examples please see the predefined DDL-DMP behavior set [DMPbaseDDL].

All properties in DDL have one or more associated behaviors. The behavior(s) of a property applies to the value of that property. It may also express constraints on other properties – typically either to the parent property (as in *maximum* above) or to the contents of a group.

A DDL *behavior* definition shall associate three pieces of information.

- A *derivation* that declares specific, more fundamental, or abstract behaviors from which this is refined by a process of specialization.
- A *description* that defines the *semantics* of the behavior – what a property with this behavior means or does.
- A *name* by which a controller can recognize that behavior.

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.

3.9.1. Semantic Descriptions

A behavior's semantic description (what it means for a property to have that behavior) is written in text and is intended primarily to be read and understood by programmers writing 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. Refinement of Behavior

All behaviors are defined as refinements of more abstract behaviors – the derivation part of a declaration. This means that definitions work from the generic to the specific – 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.

3.9.2.1. 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 the description.

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

3.9.3. Predefined Behavior

The DDL specification itself defines only the behavior NULL which applies no meaning and from which all others ultimately derive.

Each protocol using DDL may define behaviors in its specification which 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 (Section 4.8, "Declaration of of Sub Devices") other than by static inclusion.

There is a rich set of behaviors defined as part of the DMP DDL specification which may be copied or imitated for other protocols ([DMPbaseDDL]). If unified DDL descriptions are generated for equipment supporting multiple protocols, the behaviors used must be recognized across all those protocols.

3.9.4. 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 which are very different from existing ones, authors should also create intermediate behaviors in steps of refinement which seek to preserve as much generality and commonality as 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.

3.9.5. 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.

3.9.6. 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. Behaviors – 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 which 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. ddl-behaviors-dmp.html This version is not normative.

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 (Section 4.6, "String Resources") which provides a mechanism for multilingual labeling.

3.11. Property Values Types and Sizes

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. 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.

3.11.1. 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".

3.12. 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 DDL any property or included sub device can be declared to be an array and 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 properties 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.

3.12.1. 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.

3.12.2. Single vs Array Properties Within an Array

In some cases of arrays, some properties in the sub-tree will be iterated with the array while others may be single properties. For example, an array of scalar variables may share a common maximum limit. The syntax for immediate values and the (protocol dependent) address specification for network values must provide for this.

In the case of a property within an array of maximum size *N* with an immediate value, it shall specify either a single value or an array *N* values. Similarly in the case of a property within an array of maximum size *N* with a network value, the protocol interface may either specify a single address or an array of *N* addresses.

The maximum size of an array may differ from the declared size if the array is made a parameter.

A property within a two dimensional array (an array of arrays) shall either declare a number of values/addresses equal to the product of the sizes of the outer and inner arrays, or a number equal to the size of the outer array, or a single value/address.

In general, a property within an array[N_1] of arrays[N_2]... ...of arrays[Nn] may declare a number of values equal $1.N_1.N_2...Nx$ for $0 \le x \le n$.

For example, consider an array of 20 properties, each of which contains at some nested level an array of 8 sub-properties. Any sub-sub-properties (those within the 160 sub-properties) may be declared with an array of 160 values/addresses, or with an array of 20 values/addresses or with a single value/address. It may not be declared with just 8 values/addresses.

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 which are called out by this specification which makes no attempt to define or introduce XML beyond referring to these standards.

4.2.1. 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 interoperable which 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.

4.2.1.1. Encoding

DDL documents shall be encoded in Unicode encoding forms UTF-8 or UTF-16 [Unicode]. UTF-16 is indicated by the presence of a byte order mark.

4.2.1.2. Restrictions

DDL shall not contain the following XML productions:

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

4.2.1.3. XMLDecl

An XML declaration (XMLDecl) is not required. If an XML declaration is supplied it shall specify version="1.0".

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"

4.2.2. 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.

4.2.2.1. DDL Document Type Definition (DTD)

A DTD for DDL is given in Appendix 1, *A DTD for 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.

4.2.2.2. 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.

4.2.2.3. 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.

4.3. Namespace

This DDL version is identified with the namespace URL: “http://www.esta.org/acn/namespace/ddl/2005/”.

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 other uses.

4.4. Attribute and Element Value Types

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.

4.5. 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 which 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”).

4.5.1. 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), 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.

4.5.2. 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 which 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.

4.5.3. 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 which 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:

4.5.3.1. 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.

4.5.3.2. 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.

4.5.3.3. 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.

4.6. 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 which 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 which is a short name which identifies the string by lexical matching.

Strings are contained in language sets which 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.

4.6.1. 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 1. 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 which 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 2. 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 Brtitish 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 which is present in the language-set 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 which does not specify any further alternative, or will result in a loop of alternative languages which 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 which contain values for all keys in the set, and that all chains of alternate languages terminate at one such language.

4.6.2. 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 which is presenting the values and is typically a user configuration choice.

Example 3. 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 2, “languageset definition with incomplete languages”. 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”.

4.7. Element Identifiers and References

A few elements within DDL may carry an identifier (id) attribute. This is a name for the property which can be used for referencing that property from other places.

Because DDL modules may be aggregated in arbitrary ways into XML documents, it is impossible to use the XML ID/IDREF mechanism directly as the aggregation could give rise to ID clashes. The DDL mechanism is similar in principle and relies on lexical matching of the reference to an id attribute. However the scope within which matches may be made depends on the element identified and the purpose of the reference.

- A local match shall be made where the reference matches an identifier within a given device module excluding all its sub-devices, whether declared statically (using includedev) or by reference.
- A subtree match is made where the reference matches an identifier within a given device including all its sub devices.
- A global match is a subtree match against the root device in the component – that is, a match against all elements within the component including all sub devices.

  It is the responsibilty of designers to ensure that ambiguous global or subtree references do not occur. Where a device includes elements which are liable to be referenced globally it is recommended that the Identifiers on those elemnents be made parameters so that they can be changed when instantiating them in different contexts.
- The parameter mechanism uses id matches with a modification on subtree scoping in which an included paremeter whose value has been fixed is considered to have lost its id and cannot be matched – see Section 4.9, “Parametric Devices”.

It is an error for multiple elements with the same id to exist within the same local scope.

Outside of the local scope, it is not an error for multiple elements with the same id to exist, but it is an error to attempt to reference an id where multiple elements would match within the same scope.

This means for example that a device which contains an identifier may have multiple instances within a component without error, provided that that identifier is not referenced with a scope where multiple instances would match.

It is not illegal for an element which represents a repeating item in an array to hav an id, but it is illegal to reference such an element in a way which makes it ambiguous what is being referred to. For example, an property in an array which carries a **`sharedef="true"`** attribute is by definition a single property which does not iterate across the array and so can safely carry an id for reference from elsewhere. An element which is part of an array could also be referenced where the semantics of the reference allowed for the reference to apply to the entire array.

All element references within DDL refer to the instance of that element as contained in a device or component, so for example, a reference can be made to a particular property as *instantiated* within a component. This means that a reference to an external element (global scope) which is by definition not met within the local module, only becomes an error when the module is instantiated in a component which does not supply a matching id elsewhere.

4.7.1. Default Scoping of References

Unless specific rules are given for scoping of references (as with Section 4.9, “Parametric Devices”), the scope of a reference shall be as follows:

If there is a local match where the reference matches an element within the same DDL module (excluding submodules), then this is a locally scoped reference and no further matches shall be considered.

If there is no match within local scope then a globally scoped match shall apply.

If there is no match within local scope, it is an error if more than one element within the global scope matches.

It is an error if there is no match within local or global scope.

4.8. Declaration of of Sub Devices

As mentioned in Section 3.5, “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 which have different applications.

4.8.1. 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 which 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 in Section 4.5.1, “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 (Section 4.9, “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).

Figure 8. Static Inclusion Sample DDL

```
<DDL version="1.0">

  <!-- 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>
```

4.8.2. 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 which 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 which 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. (Section 4.9, “Parametric Devices”).

Figure 9. Device Reference Sample DDL

```
<DDL version="1.0">

  <!--
    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.

4.8.3. 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 itelf as a sub device or other descendant.

4.8.4. 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 which 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.

4.8.5. 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.

4.9. Parametric Devices

The rule that module content may not change (Section 4.5.1, “Module Content May not Change”) means that a processor which 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 which 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 which 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 specific property values and some other items in the description to be marked as parameters. These are called *parameter fields*.

4.9.1. Defining and Specifying Parameter Fields

A parameter field shall be either an attribute value or the content of an element which has text content. In either case the parameter value shall be a text string containing no XML markup.

The format of the value is governed by the rules provided for the individual field. Only a very restricted range of attributes or element content can be parametrized – definitions for individual elements indicate which fields may legally be made parameter fields. Attributes or element content shall not be marked as parameter fields unless explicitly identified as legal parameter fields in the rules for an individual element.

A parameter field shall be identified by first identifying the element which contains the field and then specifying the field within that element.

An element containing one or more parameter fields is called a *parametrized element*.

A parametrized element shall be marked by *preceding* it with a parameter element which shall provide a parameter identifier and a list of parameter fields.

4.9.2. Specifying Values for Parameters

The syntax of the parametrized element itself is not affected by the preceding parameter element which marks the parameters. This means that each parameter field is required to have a value provided. This value shall be used by default in any instance of the device for which an overriding value is not specified.

An overiding value for a parameter can only be specified when the device containing that parameter is included as a sub device using includedev (Section 4.8.1, "Statically Included Device Descriptions"). When a device incorporating parameters is included, values for parameters may be specified using setparam elements. Setparam identifies a parametrized element in the included device and each element within setparam identifies the field which it affects.

Figure 10. Static Inclusion With Parameters Sample DDL

```
<DDL version="1.0">

  <!-- First we define a parametric color wheel device -->

  <device UUID="c158b08d-e03e-43b3-88b2-d56dc1447155" ...>
    <label>Color Wheel</label>
    <property valuetype="network" ...>
      <label>color selector</label>
      ...
      <property valuetype="immediate" ...>
        <label>number of colors</label>
        ...
        <!-- mark the value as being a parameter -->
        <parameter id="colorCount" fields="#text"/>
        <value>6</value>
        ...
      </property>
    </property>
  </device>
  <!--
    Now we can use the definition with a new value for colorCount
  -->

  <device UUID="77ee2876-ed3a-4728-b789-2a330d55c051" ...>
    <property ...>
      <label>color changing luminaire</label>
      ...
      <includedev UUID="c158b08d-e03e-43b3-88b2-d56dc1447155">
        <setparam name="colorCount">
          <fix field="#text">8</fix>
        </setparam>
      </includedev>
      ...
    </property>
  </device>
</DDL>
```

4.9.3. Restricting Parameter Values

When a field is declared to be a parameter, 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 which is legal for that field 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 which contravenes the restriction when including the device. Restrictions are defined by choice, mininclusive, maxinclusive or refinement elements.

4.9.4. Parameters Apply Across Nested Inclusions

When a parametric device is included in a parent device description using includedev, any of its parameters which are not fixed when it is included, shall become parameters of the parent device. Overriding values may then be specified in exactly the same way if the parent is included in a grandparent description.

Parameters whose values are fixed (using fix) at the point of inclusion and all parameters within devices which are attached by reference are not parameters of the parent device and shall not be re-specified in any parent or ancestor device.

4.9.5. Adding Restrictions at Inclusion

When including a parametric device using includedev, as well as optionally fixing the values of any of its parameters, it is also allowable to add restrictions to their values (using the same restriction operators available when the parameter is originally specified) and/or to supply a new default value (see default). Any paremeter fields restricted in this way become parameters of the parent device. It is illegal to specify a parameter value which contravenes the restriction when including the device.

Any restriction value imposed on a parameter when including a device shall conform to any restriction already existing on that parameter. This means that restrictions may be made narrower by inclusion but never broader.

4.9.6. Changing Default Value at Inclusion

A parameter always has a default value. However, when a device containing a parameter is included using includedev it is possible to specify a new default value using default. This is done in much the same way as specifying an overriding value or a restriction.

A new default specified for an included parameter becomes the value used whenever the parent device occurs unless that parameter is overridden when the parent device is itself included.

4.9.7. Default Value Must Comply with Restrictions

The default value for a parameter shall always comply with any restrictions imposed on that parameter value.

This means that if restrictions are being imposed on an included parameter which would exclude the default value, then a new default which is within those restrictions must be provided at the same time.

4.9.8. Parameter Names Must Not Clash

A device shall not contain more than one parametrized element with the same ID. This applies whether the parameter is defined directly by declaration or by inclusion.

If including multiple sub devices containing parametrized elements whose names conflict, all the parameter field values within all but one of those elements must be fixed so that they do not become parameters of the parent device.

In exception to this rule, multiple inclusions of the same sub device (including arrays of the same subdevice) may have parametrized elements (whose names will necessarily be identical across all inclusions) provided that all parameters in such devices have the same values, defaults or restrictions for each inclusion.

4.9.9. 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 legal for an included device to form the entire property tree of the parent device.

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 which 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 parameter. 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.

4.10. Shared Property Mechanism

A shared property occurs where a single physical property needs to be listed in the device model in multiple places (Section 3.4.2, "Shared Properties"). For example, an single speed-limit property is provided to control noise in a motorized device. There is only one property but it needs to appear as a speed limit on each motor which it controls. The speed-limit is declared once in a "defining" declaration and is then redeclared as a “reference” property each time that it occurs.

Another example is an array of driven properties which have different targets but all share a common timer. The primary driven property is declared as an array, and its children (the target and timer driver properties) are therefore also part of the array. By making the timer shared across all items in the array, only one timer is actually present.

A shared property shall have a single fully specified defining declaration and all other occurrences of the same property have reference declarations. The defining declaration shall carry an id (except as allowed in Section 4.10.4, “Shared Properties in Arrays”) while reference declarations refer to it by the standard reference method – see Section 4.7, “Element Identifiers and References”. The defining declaration shall also carry a sharedefine attribute with the value “true”. Each reference declaration shall carry a shareref attribute which matches the id on the defining property. References follow the default scoping rules of Section 4.7.1, “Default Scoping of References”.

4.10.1. Instance Rules

These rules governing occurences of defining and reference declarations apply to instances of devices as they are combined into *appliances*. They do not apply to the device descriptions in isolation.

As shared properties use the generic property reference mechanism, the same scoping rules apply.

It is not an error if the defining declaration is not referenced (so the property is not in fact shared). This allows a property which *may* be shared to be declared in a description whether or not it is combined into an appliance with other devices which actually use it.

The defining declaration shall fully specify the property including valuetype, array size, value or network location, behavior(s) and all sub-properties.

A reference declarationshall not specify valuetype, protocol rules or sub-properties. It may provide a label and behavior(s) subject to the rules below. Labels and behaviors when present may be parametrized in the normal way.

It is an error if the element matching a shared property reference is not itself a property element.

Figure 11. Shared Properties Sample DDL

```
<DDL version="1.0">

  <!-- First define the the parent device -->

  <device UUID="c158b08d-e03e-43b3-88b2-d56dc1447155" ...>
    <label>Automated Light</label>
    ...
    <!--
        Here is the initialization property
    -->
    <property sharedefine="true"
              id="allCalibrate"
              valuetype="network" ...>
      <label>global calibration</label>
      <behavior set="example" name="initializationState"/>
      <!--
          protocol specifies the location of the property
      -->
      <protocol ...>...</protocol>
      <!--
          All sub properties appear within the defining declaration
      -->
      <property ...>...</property>
    </property>
    ...
    <!--
        include the tilt and pan sub devices
    -->
    <includedev UUID="5958e566-592c-41a8-bd70-4d1a8744e8d7"/>
    <includedev UUID="722471cc-9d7a-48a5-9a7d-46754ebdcbed"/>
  </device>

  <!--
      Here is the pan device
  -->
  <device UUID="5958e566-592c-41a8-bd70-4d1a8744e8d7" ...>
    <label>pan subdevice</label>
    ...
    <property ...>
      <label>pan axis</label>
      ...
      <!--
          The shared calibration property is an empty reference
          but has its own label (optional)
      -->
      <property shareref="allCalibrate">
        <label>Pan calibrate</label>
        <behavior set="example" name="initializationState"/>
      </property>
      ...
    </property>
  </device>

  <!--
      Tilt is similar to pan
  -->
  <device UUID="722471cc-9d7a-48a5-9a7d-46754ebdcbed" ...>
    <label>tilt subdevice</label>
    ...
    <property ...>
      <label>tilt axis</label>
      ...
      <property shareref="allCalibrate">
        <label>Tilt calibrate</label>
        <behavior set="example" name="initializationState"/>
      </property>
      ...
    </property>
  </device>

</DDL>
```

4.10.2. Behaviors of Shared Properties

To prevent the shared property mechanism from being used to impose strange or unexpected behaviors externally to a description, whenever a reference declaration has no matching defining declaration in local scope, then all behaviors of the shared property shall be declared within the reference declaration.

When a reference declaration occurs in the same device as the corresponding defining declaration (local scope), then behaviors may be declared in the reference declaration.

Wherever behaviors are declared for a shared property in accordance with these rules, the behaviors declared for the same property shall be identical for all occurrences within the appliance.

When either behavior(s) or shareref are parametrized, these rules shall apply to all instances with parameter values as they occur in an appliance.

4.10.3. Labels on Shared Properties

It is legal for different occurrences of a shared property to carry different labels. Thus in the speed limit case above, the two occurrences could be labeled X-speed and Y-speed. It is up to the application to decide how and whether to present these labels anyway, but it is recommended that when presenting labels on shared properties to a user or operator, the property be identified as shared to avoid confusion.

If both a reference declaration and the defining declaration of the same property contain a label, the label at the reference declaration should take precedence in the context of that reference.

4.10.4. Shared Properties in Arrays

The rule that there be just one defining declaration applies to the syntactic declaration in the document. It is allowable to include a defining declaration within an array property. This usage implies that there is just one property present which applies to all items in the array and in this case there may be no separate reference declarations and the defining declaration need not carry an id attribute. In the case of a network property, the protocol access declaration must also ensure that that property is at a single "location" (defnition of a location depends on the access protocol).

4.11. Logical and Syntactic Property Structures

The DDL syntax, often gives rise to properties which are declared for syntactic or structural reasons but which 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 which represent device control model at any given time, properties which govern the structure of the logical tree but which 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 which are not present in the logical tree).

Example 4. Device Reference

A device reference property has a value which 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 5. 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 which 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.

Parents

- device

- behaviorset
- languageset

5.2. altlang

attribute

Identifies the a 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 potentially part of the array (Section 3.12, "Arrays of Properties").

Value. Shall conform to xsd:unsignedInt with minInclusive set to 1.

Parents

- property
- includedev

5.3.1. Arrays of Values

When a property which 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 Section 3.12, “Arrays of Properties”.

5.4. behavior

empty element

Identifies a single behavior definition which applies to its parent property.

Parameters. The name attribute may be made a parameter field but the range of allowable values shall always be restricted using either choice or refinement.

Attributes

- name (required) the name of the behavior identified
- set (required) the UUID of the behaviorset within which the behavior definition occurs.

Parents

- property

5.5. behaviordef

element

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

Each behaviordef has a name which 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.

Every behavior is a refinement of some more generic of abstract behavior – see refines. The one exception is the behavior NULL whose UUID is "00000000-0000-0000-0000-000000000000".

The content of the behavior element is a text description of the semantics of the behavior. To allow some structuring of the prose contained within behaviordef, it is divided into sections – which 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.

Attributes

- name (required)

Children (in order)

1. label (zero or one)
2. refines (one 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 which published this behaviorset
- date (required) the publication date of this behaviorset

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

5.7. choice

element

Specifies one of a set of legal values for a parameter field (Section 4.8.1, “Statically Included Device Descriptions”). Each value within the set is specified by a separate choice element.

If a choice element is present within a setparam element then no fix, mininclusive, maxinclusive or refinement element which identifies the same field shall be present.

The value given by a choice 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 or setparam element.

A set of choices for a field shall always include the default value for the field. A default value which is within the set of choices may be specified (using default) within the same setparam element.

Attributes

- field (required) the field in the parametrized element which is to be restricted.

Content. Text content containing one value which may legally be used for the specified parameter field.

Parents

- setparam
- 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 Section 4.5, "DDL Modules – Devices, Behaviorsets and Languagesets"). Each is defined in an element of the corresponding name. To allow flexible aggregation of multiple modules into files or documents, modules are always wrapped within DDL elements.

DDL elements may be nested within DDL elements. This allows DDL documents to be included within other DDL documents without stripping outer elements. Nesting of DDL elements carries no meaning.

Note that the information content of a module is fixed and must not change once that module is assigned an identifier and published, however complete modules may be freely mixed in different combinations from one DDL *document* to another.

Attributes

- version (required)

Children (in any order)

- DDL (zero or more)
- device (zero or more)
- behaviorset (zero or more)
- languageset (zero or more)

Parents

- DDL

5.10. default

element

Specifies a value for a parameter field which shall be used in an instance of the device if no other overriding value has been specified (Section 4.8.1, "Statically Included Device Descriptions").

If a default element is present within a setparam element then no fix element which identifies the same field shall be present.

Attributes

- field (required) the field in the parametrized element which is to be restricted.

Content. Text content containing one value which may legally be used for the specified parameter field.

Parents

- setparam

5.11. device

element

The <device> element is a DDL module 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 which published this description
- date (required) the publication date of this description

Children (in order)

1. UUIDname (zero or more)
2. label (zero or one)
3. alternatefor (zero or more)
4. extends (zero or more)
5. parameter (zero or one preceding each property element)

6. Either property or includedev (one or more in any combination)

Parents

- DDL

5.12. extends

empty element

Identifies a module which 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.

Parents

- device
- behaviorset
- languageset

5.13. field

attribute

Specifies which field in a parametrized element this element relates to.

The field name shall match a name in the fields attribute of the parameter element which is being restricted or fixed.

Parents

- fix
- default

- choice
- mininclusive
- maxinclusive
- refinement

Value. The format of field shall conform to xsd:Name.

5.14. fields

attribute

A parameter field shall be either an attribute value or the content of an element which has text content. A parameter element indicates elements which contain parameter fields and which are called *parametrized elelement*s.Definitions for individual elements indicate which fields may legally be made parameter fields.

When a parameter field is an *attribute* of the parametrized element, its field name shall be the name of that attribute.

When a parameter field is the *text content* of the parametrized element, its field name shall be "#text" (without the quotes).

The fields attribute shall be a list of parameter field names, each of which is present in the parametrized element and can legally be a parameter field as specified in rules for each element.

Parents

- parameter

Value. The format of fields shall be a space separated list of field names each being a valid field name as defined in this section.

5.15. fix

element

Fixes the value of a parameter field in a sub device when that sub-device is included (Section 4.8.1, "Statically Included Device Descriptions").

If a fix element is present within a setparam element then no default, choice, mininclusive, maxinclusive or refinement element which identifies the same field shall be present.

Attributes

- field (required) the field in the parametrized element which is to be fixed.

Content. Text content containing the value to be used for the specified parameter field.

Parents

- setparam

5.16. hd

element

The heading of a section

Attributes. none

Content. Text content containing the heading.

Parents

- section

5.17. id

attribute

This attribute provides an identifier for an element which may be referenced from elsewhere. Refer to Section 4.7, “Element Identifiers and References”.

id is used for lexical matching and is case sensitive. its semantics are similar to an ID attribute defined within an XML DTD but scoping of matches may differ. An id is required to be unique within the device module within which it occurs but because of the way devices may be aggregated into XML documents, may not be unique within a document.

Value. Shall conform to xsd:Name

Parents

- parameter
- device
- includedev

5.18. includedev

element

Includedev marks the point at which a sub device is included (Section 4.8.1, “Statically Included Device Descriptions”).

Parameters

Includedev may be parametrized by preceding it with parameter and the following attributes may be parameter fields if present.

- id
- UUID

  When UUID is made a parameter field, the range of allowable values shall be restricted using choice.
- array

  When array is made a parameter field the range of allowable values shall always be restricted using either choice or maxinclusive. Further rules are given in array.

Attributes

- id (optional) an identifier for this subdevice instance.
- UUID (required) the UUID of the device being included.
- array (optional) if this is an array of sub devices.

Children (in order)

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

Parents

- device
- property

5.19. 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.

Parents

- label key identifies the string to use as the label value
- string key identifies this string

5.20. 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 which reference a string (or set of strings in different languages) which 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.

Parameters

label may be parametrized by preceding it with parameter and the following may be parameter fields. It is only meaningful to parametrize labels which occur within devices since there is no way to specify overriding values for parametrized labels within behaviorsets or languagesets.

- key
- #text content

Attributes

- key (required if no text content, forbidden otherwise) the key identifying a string or strings.
- set (required if no text content, forbidden otherwise) the UUID of the languageset in which the string(s) may be found.

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

Parents

- behaviordef
- behaviorset
- device
- languageset
- property

5.21. 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

5.22. language

element

A language contains a set of string definitions in a particular language which is identified by its lang attribute. It may optionally specify an alternative language within the same languageset which 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)

Children

- string (one or more)

Parents

- languageset

5.23. languageset

element

A languageset is a container for string resources with variants in multiple languages (Section 4.6, "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 which is used by elements which 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 which published this languageset
- date (required) the publication date of this languageset

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

5.24. maxinclusive

element

Specifies a maximum value restriction for a numeric parameter field. (Section 4.8.1, “Statically Included Device Descriptions”).

If a maxinclusive element is present within a setparam or parameter element then no fix, choice or refinement element which identifies the same field shall be present.

maxinclusive shall only be used for a parameter field which has a numeric value.

The value given by a maxinclusive 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 or setparam element.

A maxinclusive value for a field shall never be less than the default value for the field. A default value which is complies with this may be specified (using default) within the same setparam element.

Attributes

- field (required) the field in the parametrized element which is to be restricted.

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

Parents

- setparam
- parameter

5.25. mininclusive

element

Specifies a minimum value restriction for a numeric parameter field. (Section 4.8.1, “Statically Included Device Descriptions”).

If a mininclusive element is present within a setparam or parameter element then no fix, choice or refinement element which identifies the same field shall be present.

mininclusive shall only be used for a parameter field which has a numeric value.

The value given by a mininclusive 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 or setparam element.

A mininclusive value for a field shall never be greater than the default value for the field. A default value which is complies with this may be specified (using default) within the same setparam element.

Attributes

- field (required) the field in the parametrized element which is to be restricted.

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

Parents

- setparam
- parameter

5.26. name

attribute

A name which 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 which exists in a different context.

Value. Shall conform to xsd:NCName

Parents

- behavior
- behaviordef
- device
- protocol
- refines
- useprotocol
- UUIDname

5.27. 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

Content. Text content is the text of the paragraph.

Parents

- section

5.28. parameter

element

<parameter> shall be used to specify that the *following sibling* element contains parametric fields.

<parameter> provides an identifier for the element and a list of parameter fields. It may optionally contain elements restricting the range of values the parameter fields may take.

Attributes

- id (required) the identifier for this parameter
- fields (required) list of fields to be treated as parameters

Children (in any order)

- mininclusive (zero or more)
- maxinclusive (zero or more)
- refinement (zero or more)
- choice (zero or more)

Parents

- device
- property
- protocol and other elements within protocol as specified in rules for individual *access protocols*

May Precede

- behavior
- label
- property
- value
- protocol specific elements as defined by individual protocol rules

5.29. 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.

A shared property declaration shall either be a defining declaration or a reference declaration (see Section 4.10, “Shared Property Mechanism”).

Parameters

Property may be parametrized by preceding it with parameter and the following attributes may be parameter fields if present.

- id
- shareref
- sharedefine
- array

  When array is made a parameter field the range of allowable values shall always be restricted using either choice or maxinclusive. Further rules are given in array.

Parents

- device
- property

Other syntax depends on whether the property is shared and whether it is a defining or reference declaration:

5.29.1. Unshared Property or Defining Declaration of Shared Property

Attributes

- array (optional) if not present defaults to 1 (a single value).
- id (optional)
- valuetype (required)
- sharedefine
  - optional if an unshared property.
  - required if a defining declaration.

Children (in order):

1. label (optional)
2. behavior (one or more)
3. value (one or more if valuetype="immediate", forbidden otherwise). See Section 3.12, “Arrays of Properties” for number of occurrences.
4. protocol (required if valuetype="network", optional otherwise).
5. Either property or includedev sub properties:
    - one or more in any combination if valuetype="NULL" or "implied"
    - zero or more in any combination if valuetype=“network” or “immediate”

5.29.2. Shared Property Reference Declaration

Attributes

- id (optional)
- shareref (required). Identifies that this is the same property as all others in the appliance with a matching value.

Children (in order):

1. label (optional)
2. behavior (zero or more) (one or more if defining declaration is not in the same device).

5.30. 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 which 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.

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

Parents

- property
- includedev

5.31. 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 which clearly identifies the organization and the appropriate department or section.

Value. Should conform to xsd:anyURI

Parents

- device
- behaviorset
- languageset

5.32. refinement

element

Shall only be used when the parameter field identified is a behavior name. <refinement> specifies a base behavior for the parameter. Any value set for the parameter field (using default, choice or fix) shall be a refinement of this base behavior.

If a refinement element is present within a setparam or parameter element then no fix or choice element which identifies the same field shall be present. (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 or setparam element.

A refinement value for a field shall never exclude the default value for the field. A default value which is complies with this may be specified (using default) within the same setparam element.

Attributes

- field (shall be present with the value "name") identifies the field in the parametrized element which is to be restricted.

Content. Text content shall conform to xsd:NCName.

Parents

- setparam
- parameter

5.33. refines

empty element

Every behavior definition (except "NULL") must be a refinement of one or more existing definitions. Refines identifies a single behavior which the current behaviordef derives form.

Attributes

- name (required) the name of the behavior identified
- set (required) the UUID of the behaviorset within which the behavior definition occurs.

Parents

- behaviordef

5.34. section

element

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

Attributes. none

Children (in order)

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

Parents

- behaviordef
- section

5.35. 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

5.36. setparam

element

Specifies the value a parameter is to take, when a sub device is included in a description (Section 4.8.1, "Statically Included Device Descriptions"). For each field in the parametrized element, either a fixed value, a new default value or a restriction may be specified.

Attributes

- name (required) the key identifying this string.

Children (in any order)

- fix (zero or more)
- default (zero or more)
- mininclusive (zero or more)
- maxinclusive (zero or more)
- refinement (zero or more)

Parents

- includedev

5.37. sharedefine

attribute

Identifies the defining declaration of a shared property (see Section 3.4.2, “Shared Properties”, Section 4.10, “Shared Property Mechanism” and property).

Value. Shall conform to xsd:boolean.

Parents

- property

See also. shareref

5.38. shareref

attribute

This attribute connects reference declarations of a shared property to their defining declaration. All reference declarations of the same shared property within an appliance shall have the same value for shareref. (see Section 3.4.2, “Shared Properties”, Section 4.10, “Shared Property Mechanism” and property).

shareref is used for lexical matching to an id attribute and is case sensitive.

Value. Shall conform to xsd:Name

Parents

- property

See also. sharedefine, id

5.39. 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.

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

Parents

- language

5.40. #text

text content

The name `#text` is used to identify the text content of an element when specifying fileds which are parameters. See field and fields attributes.

5.41. 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 which 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 6. 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

5.42. useprotocol

empty element

Identifies a protocol which must be used to access a device.

Attributes

- name (required) shall match an ESTA defined protocol name [ESTA-IDs].

Parents

- device

5.43. UUID

attribute

A UUID attribute shall either contain a literal UUID value or a UUID name which has been assigned using the UUIDname element and which 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 for device modules are also called *DCIDs*.

> **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 7. 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
- behaviorset
- languageset

- UUIDname
- alternatefor
- extends

5.44. UUIDname

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 – the UUID to be assigned the name. Shall be specified explicitly Named UUIDs are not permitted.
- name Required – shall be 32 characters or shorter.

Content. Empty

Parents

- device
- behaviorset
- languageset

5.45. 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 have multiple values according to the rules in Section 3.12, “Arrays of Properties”.

Parameters. Value may be parametrized by preceding it with parameter. The #text content is the only field permitted.

Attributes

- type (required) the type of value contained

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

Parents

- property

5.46. valuetype

attribute

Specifies the value type of a property.

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

implied
: The property has a hidden value which is driven by other properties (driven properties).

Parents

- property

5.47. version

attribute

This is the DDL version. The version number “1.0” shall be used to indicate conformance to this version of this specification; it is an error for a document to use the value “1.0” 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. Value shall be “1.0”

Parents

- DDL

5.48. xml:space

attribute

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

6. Summary of Parametrizable Elements and Fields

The following fields may be specified as parameters in DDL.

- label element
  - #text content (when label is literal)
  - key attribute (when label is string reference)
- property element

  - shareref attribute
  - id attribute
  - array attribute
- value element
  - #text content
- behavior element
  - name attribute
- includedev element
  - id attribute

7. 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] which defines the interface to a real protocol.

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

7.1. Variable Access

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 which 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.

7.1.1. 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 which are required to access and handle the data without necessarily interpreting it.

7.2. Necessary Definitions and Restrictions

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

7.2.1. 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].

7.2.2. Variable 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.

7.3. Behaviors

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

1. A DTD for DDL

1.1. Generic DDL DTD

This DTD (refer to [XML]]) is provided for reference purposes. See Section 4.2.2, “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 1.0//EN**”

```
<!--
  DTD for Device Descritpion Language

  No protocol declared
-->

<!ELEMENT DDL ((DDL | behaviorset | device | languageset)*)>
<!ATTLIST DDL
  version CDATA #FIXED "1.0"
>

<!-- common module content -->

<!ELEMENT label (#PCDATA)>
<!ATTLIST label
  set NMTOKEN #IMPLIED
  key NMTOKEN #IMPLIED
>

<!ELEMENT alternatefor EMPTY>
<!ATTLIST alternatefor
```

```
  UUID NMTOKEN #REQUIRED
>

<!ELEMENT extends EMPTY>
<!ATTLIST extends
  UUID NMTOKEN #REQUIRED
>

<!ELEMENT UUIDname EMPTY>
<!ATTLIST UUIDname
  name NMTOKEN #REQUIRED
  UUID NMTOKEN #REQUIRED
>

<!-- languageset module -->

<!ELEMENT languageset (
  UUIDname*, label?,
  (alternatefor | extends)*,
  language+
)>
<!ATTLIST languageset
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
>

<!-- languageset content -->

<!ELEMENT language (string+)>
<!ATTLIST language
  lang CDATA #REQUIRED
  altlang CDATA #IMPLIED
>

<!ELEMENT string (#PCDATA)>
<!ATTLIST string
  key NMTOKEN #REQUIRED
>

<!-- behaviorset module -->

<!ELEMENT behaviorset (
  UUIDname*, label?,
  (alternatefor | extends)*,
  behaviordef+
)>
<!ATTLIST behaviorset
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
>

<!-- behaviorset content -->

<!ELEMENT behaviordef (label, refines+, section+)>
```

```
<!ATTLIST behaviordef
  name NMTOKEN #REQUIRED
>

<!ELEMENT refines EMPTY>
<!ATTLIST refines
  set NMTOKEN #REQUIRED
  name NMTOKEN #REQUIRED
>

<!ELEMENT section (hd?, (section | p)+)>

<!ELEMENT hd (#PCDATA)>

<!ELEMENT p (#PCDATA)>
<!ATTLIST p xml:space (default | preserve) 'default'>

<!-- device module -->

<!ELEMENT device (
  UUIDname*, (parameter?, label)?,
  (alternatefor | extends)*,
  useprotocol,
  (parameter?, (property | includedev))+
)>
<!ATTLIST device
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
>

<!-- device content -->

<!ELEMENT useprotocol EMPTY>
<!ATTLIST useprotocol
  name NMTOKEN #REQUIRED
>

<!ELEMENT property (
  ( parameter
  | label
  | behavior
  | value
  | protocol
  | property
  | includedev)*
)>
<!ATTLIST property
  id NMTOKEN #IMPLIED
  sharedefine (true | false) "false"
  shareref NMTOKEN #IMPLIED
  array CDATA #IMPLIED
  valuetype (NULL | immediate | implied | network) #IMPLIED
>

<!ELEMENT behavior EMPTY>
```

```
<!ATTLIST behavior
  set NMTOKEN #REQUIRED
  name NMTOKEN #REQUIRED
>

<!ELEMENT value (#PCDATA)>
<!ATTLIST value
  type (uint | sint | float | string | object) #REQUIRED
>

<!ELEMENT protocol ANY >
<!ATTLIST protocol
  name CDATA #REQUIRED
>

<!-- parameter declarations -->

<!ELEMENT parameter
        (choice | mininclusive | maxinclusive | refinement)* >
<!ATTLIST parameter
  id NMTOKEN #REQUIRED
  fields CDATA #REQUIRED
>

<!-- included devices -->

<!ELEMENT includedev ( setparam* )>
<!ATTLIST includedev
  id NMTOKEN #IMPLIED
  UUID NMTOKEN #REQUIRED
>

<!ELEMENT setparam
        (
          fix | default | choice
          | mininclusive | maxinclusive | refinement
        )+ >
<!ATTLIST setparam
  name NMTOKEN #REQUIRED
>

<!ELEMENT fix (#PCDATA)>
<!ATTLIST fix
  field CDATA #REQUIRED
>

<!ELEMENT default (#PCDATA)>
<!ATTLIST default
  field CDATA #REQUIRED
>

<!ELEMENT choice (#PCDATA)>
<!ATTLIST choice
  field CDATA #REQUIRED
>
```

```
<!ELEMENT mininclusive (#PCDATA)>
<!ATTLIST mininclusive
  field CDATA #REQUIRED
>

<!ELEMENT maxinclusive (#PCDATA)>
<!ATTLIST maxinclusive
  field CDATA #REQUIRED
>

<!ELEMENT refinement (#PCDATA)>
<!ATTLIST refinement
  field CDATA #REQUIRED
>
```

2. 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].

2.1. Relationship Between DDL Devices and DMP Components

Each DMP *component* (see [Arch]) which exposes properties shall form a single *appliance*. The UUID [UUID] of its single root device shall 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.

2.2. 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.

2.3. 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 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.

2.3.1. Addressing

DMP properties are single values which have a one-to-one correspondence with DDL Network Accessed properties (Network property). A DMP property is specified using a single scalar address range (0..42949672695).

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

2.3.1.1. Relative Addressing and Location Origin

To provide the address of a DMP property, a relative addressing scheme is available in DDL and is the default address syntax. DMP addresses or network properties are expressed as signed offsets relative to a location origin which is specified 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 which are relative to the location of the root of the subtree. This allows a subtree to be repeated elsewhere in the address space with 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.

2.3.1.2. 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 prevents re-use of devices by inclusion or reference.

2.3.1.3. Virtual Addressing

The ability of a device to support virtual addressing of a property is indicated by the virtual attribute. See [DMP] for details of virtual addressing. This attribute has no affect on the other addressing attributes of the property since virtual addressing exists in addition to actual addressing.

2.3.1.4. Non-network Properties

Properties which 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.

2.3.2. Property Arrays

In accordance with Section 3.12, “Arrays of Properties” any property may be declared as an array. A property which is declared as an array or is a descendant of an array property shall either be shared or shall have one value per array item at different DMP addresses. Where a DMP network property is iterated through an array, it shall declare an increment which specifies the increment from one item of the array to the next. It may also specify a separate increment for its children using childrule_DMP.

2.3.3. 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 Set_Property and 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.

2.3.4. 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 which have a fixed format.

The value of any particular DMP property shall 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.

2.3.5. 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 DMP behavior set.

2.3.6. Shared Properties in DMP

In accordance with the rules given for shared property syntax in any DDL (see Section 4.10, "Shared Property Mechanism"), the following rules apply to DMP specific elements.

In a defining declaration of a shared property which is an array or is contained in an array, any DMP address specified shall resolve to the same address for all items in the array. Normally this means that inc must be zero, but in the case of a sub property whose location origin changes through the array, inc must be set to cancel that change.

In a defining declaration of a shared property which is an array or is contained in an array, any DMP address specified shall resolve to the same address for all items in the array. Normally this means that inc must be zero, but in the case of a sub property whose location origin changes through the array, inc must be set to cancel that change.

2.3.7. Byte Ordering

While specific encodings are described in DDL behaviors, it is forbidden to create behaviors which 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.

2.4. DMP Element and Attribute Reference

2.4.1. protocol (DMP devices)

element

In DMP a protocol element shall have the following content.

Attributes

- name (required) shall have the value “ESTA.DMP”

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

2.4.2. propref_DMP

empty element

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

Parameters. propref_DMP may be parametrized by preceding it with parameter and *all* attributes present may be parameter fields.

Attributes

- loc (required)
- size (required)
- abs (optional)
- virtual (optional)

- read (optional)
- write (optional)
- event (optional)
- varsize (optional)
- inc (optional)

Parents

- protocol

2.4.3. 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 shall inherit its own location origin unchanged.

The location origin for the root properties in a device which is attached by reference (see Section 4.8.2, “Sub Devices Attached by Reference”) shall be received from the device-reference property Section 4.8, “Declaration of of Sub Devices” which 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 which is statically included (see Section 4.8.1, “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 properies of the root device in an *appliance* shall be 0.

Parameters. childrule_DMP may be parametrized by preceding it with parameter and *all* attributes present may be parameter fields.

Attributes

- loc (optional)
- abs (optional if loc is present, forbidden otherwise)
- inc (optional)

Parents

- protocol

2.4.4. 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 Section 2.3.1.1, “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

2.4.5. 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.

Parents

- propref_DMP

- childrule_DMP

2.4.6. 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

Parents

- propref_DMP
- childrule_DMP

2.4.7. virtual

attribute

If true then this property may be accessed using virtual addressing in DMP. See [DMP] for details of how to set up and use virtual addresses.

Value. Shall conform to xsd:boolean.

Parents

- propref_DMP

2.4.8. 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.

Parents

- propref_DMP

2.4.9. 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.

Parents

- propref_DMP

2.4.10. 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 for properties which 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.

Parents

- propref_DMP

2.4.11. 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 Section 2.3.4, “Property size and variable size properties”.

Value. Shall conform to xsd:boolean.

Parents

- propref_DMP

2.4.12. 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 Section 2.3.4, “Property size and variable size properties”.

Value. Shall conform to xsd:unsignedShort

Parents

- propref_DMP

2.5. Summary of Parametrizable Elements and Fields for DMP (informative)

The following fields may be defined as parameters in DDL for DMP.

- label element
  - #text content (when label is literal)
  - key attribute (when label is string reference)
- property element
  - shareref attribute
  - id attribute
  - array attribute
- value element
  - #text content
- behavior element
  - name attribute
- includedev element
  - id attribute
- propref_DMP element
  - loc attribute
  - abs attribute
  - inc attribute

  - virtual attribute
  - read attribute
  - write attribute
  - event attribute
  - size attribute
  - varsize attribute
- childloc_DMP element
  - loc attribute
  - abs attribute
  - inc attribute

2.6. Examples

These assume that an (non-existent) example behaviorset “f3347bac-3e20-437d-a69e-019f37a71288” has been assigned the name “foo” in the parent device, and that the dmp core behaviorset “a713a314-a14d-11d9-9f34-000d613667e2” has been assigned the name “dmp” (using UUIDname):

```
<device ...>
  <UUIDname UUID="f3347bac-3e20-437d-a69e-019f37a71288" name="foo"/>
  <UUIDname UUID="a713a314-a14d-11d9-9f34-000d613667e2" name="dmp"/>
  ...
</device>
```

2.6.1. Simple Property

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

Figure 2.1. 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>
```

2.6.2. 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 through 511 within the DMP address space:

Figure 2.2. 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>
```

2.6.3. 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):

Figure 2.3. DDL for a Direct Device Reference

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

2.6.4. 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.

Figure 2.4. DDL for a Direct Device Reference

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

2.6.5. 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 could be notified should the type of the sub-device be changed dynamically.

Figure 2.5. DDL for an External Device Reference

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

2.6.6. 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” field of the “bar” parameter for all subdevices is set to “99”.

Figure 2.6. DDL for an Array of Direct Device References

```
<property valuetype="NULL" array="96">
  <behavior name="group" set="dmp"/>

  <includedev
    UUID="38763d0c-b0f9-11d9-8615-000d613667e2"
    array="96"
  >
    <protocol name="ESTA.DMP">
      <childrule_DMP
        loc="500"
        abs="true"
        inc="1000"
      />
    </protocol>
    <setparam name="bar">
      <fix field="foo">99</fix>
    </setparam>
  </includedev>
</property>
```

2.6.7. 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.

Figure 2.7. DDL for an Array of External Device References

```
<property valuetype="network" array="16">
  <behavior name="deviceRef" set="dmp"/>
  <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 2.1. 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.. 4294967296	Available for other properties or sub devices

2.6.8. 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 which 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 has an inc of 0 indicating that it does not increment with the HTP inputs 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.

Figure 2.8. DDL for an Array of Direct Device References

```
<property array="96" valuetype="network">
  <label>Dimmer Array</label>
  <behavior name="intensity" set="foo" />
  <behavior name="HTPdriven" set="foo" />
  <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" />
    <protocol name="ESTA.DMP">
      <propref_DMP inc="1" loc="0"
                   read="true" size="2" write="true" />
    </protocol>

    <property valuetype="network">
```

```
      <label>Preheat</label>
      <behavior name="minLimit" set="foo" />
      <protocol name="ESTA.DMP">
        <propref_DMP inc="0" loc="19"
                     read="true" size="2" write="true" />
      </protocol>
    </property>
  </property>
</property>
```

3. Schemas for DMP Use of DDL

3.1. 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 1.0//EN//DMP`**"

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

  Registered protocol name is "ESTA.DMP"
-->

<!ELEMENT DDL ((DDL | behaviorset | device | languageset)*)>
<!ATTLIST DDL
  version CDATA #FIXED "1.0"
>

<!-- common module content -->

<!ELEMENT label (#PCDATA)>
<!ATTLIST label
  set NMTOKEN #IMPLIED
  key NMTOKEN #IMPLIED
>

<!ELEMENT alternatefor EMPTY>
<!ATTLIST alternatefor
  UUID NMTOKEN #REQUIRED
>

<!ELEMENT extends EMPTY>
<!ATTLIST extends
  UUID NMTOKEN #REQUIRED
>
```

```
<!ELEMENT UUIDname EMPTY>
<!ATTLIST UUIDname
  name NMTOKEN #REQUIRED
  UUID NMTOKEN #REQUIRED
>

<!-- languageset module -->

<!ELEMENT languageset (
  UUIDname*, label?,
  (alternatefor | extends)*,
  language+
)>
<!ATTLIST languageset
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
>

<!-- languageset content -->

<!ELEMENT language (string+)>
<!ATTLIST language
  lang CDATA #REQUIRED
  altlang CDATA #IMPLIED
>

<!ELEMENT string (#PCDATA)>
<!ATTLIST string
  key NMTOKEN #REQUIRED
>

<!-- behaviorset module -->

<!ELEMENT behaviorset (
  UUIDname*, label?,
  (alternatefor | extends)*,
  behaviordef+
)>
<!ATTLIST behaviorset
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
>

<!-- behaviorset content -->

<!ELEMENT behaviordef (label, refines+, section+)>
<!ATTLIST behaviordef
  name NMTOKEN #REQUIRED
>

<!ELEMENT refines EMPTY>
<!ATTLIST refines
  set NMTOKEN #REQUIRED
  name NMTOKEN #REQUIRED
```

```
>

<!ELEMENT section (hd?, (section | p)+)>

<!ELEMENT hd (#PCDATA)>

<!ELEMENT p (#PCDATA)>
<!ATTLIST p xml:space (default | preserve) 'default'>

<!-- device module (DMP variant: useprotocol is fixed) -->

<!ELEMENT device (
  UUIDname*, (parameter?, label)?,
  (alternatefor | extends)*,
  useprotocol,
  (parameter?, (property | includedev))+
)>
<!ATTLIST device
  UUID NMTOKEN #REQUIRED
  provider CDATA #REQUIRED
  date NMTOKEN #REQUIRED
>

<!-- device content -->

<!-- useprotocol fixed for DMP -->
<!ELEMENT useprotocol EMPTY>
<!ATTLIST useprotocol
  name NMTOKEN #FIXED "ESTA.DMP"
>

<!ELEMENT property (
  ( parameter
  | label
  | behavior
  | value
  | protocol
  | property
  | includedev)*
)>
<!ATTLIST property
  id NMTOKEN #IMPLIED
  sharedefine (true | false) "false"
  shareref NMTOKEN #IMPLIED
  array CDATA #IMPLIED
  valuetype (NULL | immediate | implied | network) #IMPLIED
>

<!ELEMENT behavior EMPTY>
<!ATTLIST behavior
  set NMTOKEN #REQUIRED
  name NMTOKEN #REQUIRED
>

<!ELEMENT value (#PCDATA)>
<!ATTLIST value
```

```
  type (uint | sint | float | string | object) #REQUIRED
>

<!-- Protocol dependent section for DMP -->

<!ELEMENT protocol (propref_DMP?, childrule_DMP?) >
<!ATTLIST protocol
  name CDATA #FIXED "ESTA.DMP"
>

<!ELEMENT propref_DMP EMPTY>
<!ATTLIST propref_DMP
  loc CDATA #REQUIRED
  abs (true | false) "false"
  inc CDATA #IMPLIED
  virtual (true | false) "false"
  size CDATA #REQUIRED
  read (true | false) "false"
  write (true | false) "false"
  event (true | false) "false"
  varsize (true | false) "false"
>

<!ELEMENT childrule_DMP EMPTY>
<!ATTLIST childrule_DMP
  loc CDATA #REQUIRED
  abs (true | false) "false"
  inc CDATA #IMPLIED
>
<!-- End of protocol dependent section -->

<!-- parameter declarations -->

<!ELEMENT parameter
        (choice | mininclusive | maxinclusive | refinement)* >
<!ATTLIST parameter
  id NMTOKEN #REQUIRED
  fields CDATA #REQUIRED
>

<!-- included devices -->

<!ELEMENT includedev ( setparam* )>
<!ATTLIST includedev
  id NMTOKEN #IMPLIED
  UUID NMTOKEN #REQUIRED
>

<!ELEMENT setparam
        (
          fix | default | choice
          | mininclusive | maxinclusive | refinement
        )+ >
<!ATTLIST setparam
  name NMTOKEN #REQUIRED
>
```

```
<!ELEMENT fix (#PCDATA)>
<!ATTLIST fix
  field CDATA #REQUIRED
>

<!ELEMENT default (#PCDATA)>
<!ATTLIST default
  field CDATA #REQUIRED
>

<!ELEMENT choice (#PCDATA)>
<!ATTLIST choice
  field CDATA #REQUIRED
>

<!ELEMENT mininclusive (#PCDATA)>
<!ATTLIST mininclusive
  field CDATA #REQUIRED
>

<!ELEMENT maxinclusive (#PCDATA)>
<!ATTLIST maxinclusive
  field CDATA #REQUIRED
>

<!ELEMENT refinement (#PCDATA)>
<!ATTLIST refinement
  field CDATA #REQUIRED
>
```

3.2. RELAX NG Schema for DMP specific DDL

This schema uses the compact form of RELAX NG schema language [RELAXNG]] together with XML Schema data typing [XSD].

RELAX NG does not include an internal mechanism for associating schemas with documents, however, where a public name for this schema is required the name shall be "**-//ESTA//RELAXNG Device Description Language 1.0//EN//DMP**"

```
namespace dtd = "http://relaxng.org/ns/compatibility/annotations/1.0"

# datatypeLibrary="http://relaxng.org/ns/compatibility/datatypes/1.0"

#
# Relax NG schema for DDL applied to DMP protocol
#

protocolName = "ESTA.DMP"

start = DDL

DDL =
  element DDL { DDL_atts, (DDL | module)* }
```

```
DDL_atts = attribute version { "1.0" }

#
# Common to all modules
#

module = languageset | behaviorset | device

module_atts =
  attribute UUID { xsd:NMTOKEN },
  attribute provider { text },
  attribute date { xsd:date }

related_module = alternatefor | extends

module_header = module_atts, UUIDname*, label?, related_module*
device_module_header =
    module_atts,
    UUIDname*,
    (parameter?, label)?,
    related_module*

UUIDname =
  element UUIDname {
    attribute name { xsd:NCName },
    attribute UUID { xsd:NMTOKEN }
  }

alternatefor =
  element alternatefor {
    attribute UUID { xsd:NMTOKEN }
  }

extends =
  element extends {
    attribute UUID { xsd:NMTOKEN }
  }

label =
  ( label_literal | label_reference )


label_literal =
  element label {
    text
  }

label_reference =
  element label {
    attribute set { xsd:NMTOKEN },
    attribute key { xsd:NCName }
  }

#
# Languageset
```

```
#

languageset =
  element languageset {
    module_header,
    language+
  }

language =
  element language {
    attribute lang { xsd:language },
    attribute altlang { xsd:language }?,
    \string+
  }

\string =
  element string {
    attribute key { xsd:NCName },
    text
  }

#
# Behaviorset
#

behaviorset =
  element behaviorset {
    module_header,
    behaviordef+
  }

behaviordef =
  element behaviordef {
    attribute name { xsd:NCName },
    label,
    refines+,
    section+
  }

refines =
  element refines {
    attribute set { xsd:NMTOKEN },
    attribute name { xsd:NCName }
  }

section =
  element section {
    hd?,
    (section | p)+
  }

hd =
  element hd { text }

p =
  element p {
    attribute xml:space { "default" | "preserve" }?,
    text
  }

#
# device
#

device =
  element device {
    device_module_header,
    useprotocol,
    ( (parameter?, property) | includedev )+
  }

useprotocol =
  element useprotocol {
    attribute name { protocolName }
  }

#
# Included devices
#

includedev =
  element includedev {
    attribute id { xsd:Name },
    attribute UUID { xsd:NMTOKEN },
    attribute array { xsd:positiveInteger }?,
    protocol_childRules?,
    setparam*
  }

#
# Property
#

property =
  element property {
    attribute id { xsd:Name },
    ((
      attribute shareref { xsd:Name },
      propRefContent
    ) | (
      attribute sharedefine { xsd:boolean }?,
      propDefineContent
    ))
  }

propRefContent = (
    (parameter?, label)?,
    (parameter?, behavior)*
  )

propDefineContent = (
    attribute array { xsd:positiveInteger }?,
    (parameter?, label)?,
    (parameter?, behavior)+,
    (nullPropVal | impliedPropVal | netPropVal | immPropVal),
    (parameter?, (property | includedev))*
  )

nullPropVal =
  attribute valuetype { "NULL" },
  protocol_childRules?

impliedPropVal =
  attribute valuetype { "implied" },
  protocol_childRules?

netPropVal =
  attribute valuetype { "network" },
  protocol_propertyRef

immPropVal =
  attribute valuetype { "immediate" },
  (parameter?, value)+,
  protocol_childRules?

behavior =
  element behavior {
    attribute name { xsd:NCName },
    attribute set { xsd:NMTOKEN }
  }

value =
  element value {
    attribute type {
      "uint" | "sint" | "float" | "string" | "object"
    },
    text
  }

#
# Parameter declaration
#

parameter =
  element parameter {
    attribute id { xsd:Name },
    attribute fields {
      list {
      "#text"?,
      "array"?,
      "key"?,
      "id"?,
      "sharedefine"?,
      "shareref"?,
      "name"?,
      "UUID"?,
      "loc"?,
      "abs"?,
      "inc"?,
      "virtual"?,
      "read"?,
      "write"?,
      "event"?,
      "size"?,
      "varsize"?
      }
    },
    ( choice | mininclusive | maxinclusive | refinement )*
  }

#
# Parameter instantiation
#

setparam =
  element setparam {
    attribute name { xsd:NCName },
    ( fix | \default | choice
      | mininclusive | maxinclusive | refinement )+
  }

restrictparam =
  attribute field {
      "#text"
      | "array"
      | "key"
      | "id"
      | "sharedefine"
      | "shareref"
      | "name"
      | "UUID"
      | "loc"
      | "abs"
      | "inc"
      | "virtual"
      | "read"
      | "write"
      | "event"
      | "size"
      | "varsize"
  },
  text

fix = element fix { restrictparam }

\default = element default { restrictparam }

choice = element choice { restrictparam }

mininclusive = element mininclusive { restrictparam }

maxinclusive = element maxinclusive { restrictparam }

refinement = element refinement { restrictparam }
```

```
#
# Protcol dependent structures
#
# The protocol element can occur in any property
# However, it can only define a property reference
# when it occurs in a Network type property. In NULL
# immediate and implied properties the protocol
# can only apply rules for child properties and
# their references
#

protocol_propertyRef =
  element protocol {
    attribute name { protocolName },
    (parameter?, propref_DMP),
    (parameter?, childrule_DMP)?
  }

protocol_childRules =
  element protocol {
    attribute name { protocolName },
    (parameter?, childrule_DMP)
  }

DMP_location =
    attribute loc { xsd:int }?,
    attribute abs { xsd:boolean }?,
    attribute inc { xsd:short }?,
    attribute virtual { xsd:boolean }?

DMP_access =
    attribute size { xsd:unsignedShort }?,
    attribute read { xsd:boolean }?,
    attribute write { xsd:boolean }?,
    attribute event { xsd:boolean }?,
    attribute varsize { xsd:boolean }?

propref_DMP =
  element propref_DMP { DMP_location & DMP_access }

childrule_DMP =
  element childrule_DMP { DMP_location }
```

4. Languagesets and Behaviors Defined for DDL Operation with DMP

A base set of property behaviors is defined for use with DMP. These and associated rules are specified in [DMPbaseDDL]. These have the UUID a713a314-a14d-11d9-9f34-000d613667e2 and 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). Labels for these behaviors are contained in languageset 98757e30-a14e-11d9-8fcc-000d613667e2.

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.

Definitions

access protocol	The network or datalink protocol used to access and control the devices described by a Device Description (e.g. DMP for a DMP device). DDL may be adapted to many different access protocols both within and outside of E1.17 and a single device may support multiple access protocols.
appliance	In DDL an appliance is a piece of equipment described by a root device and all its children and descendents. In DMP systems an appliance corresponds to a component which exposes one or more devices (since the rules require that all devices are descendants of a single root device). See Also root device.
component	The process, program or application corresponding to a single ACN endpoint. All messages in ACN are sent and received by a component. See [Arch] for a more complete definition.
controller (DDL)	A machine or program which is used to control devices described by DDL. The controller is connected to the device(s) via one or more data links or networks using one or more access protocols.
DCID	Device class identifier. This is a unique identifier *UUID* for a Device Class. All devices have a DCID which 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)	A device is an entity which may be monitored and controlled by means of a network or datalink and where a DDL description is available describing how to do this. A device includes all those devices (sub-devices) contained within it (device is a recursive term).
device class	The set of devices which are all described by the same device description.
root device	An instance of a device described in DDL which has no parent device. See Also appliance.

semantic	A term common in linguistics, computer science, logic and formal languages (e.g. DDL) which 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).

References

Normative

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

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

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

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

[DMPbaseDDL] Entertainment Services and Technology Association [http://www.esta.org/tsp/]. ESTA TSP CP/2005-1014R0. *Entertainment Technology - Architecture for Control Networks*. EPI 22. Base DDL Modules for DMP Devices. 2006-10-19.

[ESTA-IDs] Entertainment Services and Technology Association [http://www.esta.org/tsp/]. ESTA TSP CP/2004-1027R3. *Entertainment Technology - Architecture for Control Networks*. EPI-16 ESTA Registered Names and Identifiers – Format and Procedure for Registration. 2006-10-19.

[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)

[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 (Third Edition)*. 1.0 (Second Edition). W3C Recommendation. 4 February 2004. http://www.w3.org/TR/2004/REC-xml-20040204/.

[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/.

Informative

[DMX512] Entertainment Services and Technology Association [http://www.esta.org/tsp/]. ANSI E1.11-2004. *USITT DMX512-A - Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories*. 2004.

[RDM] Entertainment Services and Technology Association [http://www.esta.org/tsp/]. ANSI E1.20-2006. *Entertainment Technology - Remote Device Management over DMX512 Networks*. 2006.