

**MOBILE AND PERVASIVE COMPUTING LABORATORY  
UNIVERSITY OF FLORIDA**

**Device Description Language  
Specification  
(Version 1.2)**

**5 November 2008**

## Revision History

Document version	Date	Contributor	Summary of additions/changes
1.2.2	Sept 18, 2008	Chao Chen	1. Added Appendix C.V. <i>Sample DDL Documents for AnD Blood Pressure Monitor</i> .
1.2.1	Sept.10, 2008	Chao Chen	1. Name of the language changed to DDL. 2. Re-formatted. 3. Updated data models in Appendix A.
1.2.0	July 10, 2008	Chao Chen Hen-I Yang	1. Added support for complex devices. 2. Added support for safety properties and emergency handlers. 3. Added section <i>Safety Mechanisms for Devices</i> . 4. <i>Revision history table</i> moved to the front. 5. Revised the graph for the DDL device driver generator application. 6. Added section 1.6 <i>DDL Applications</i> . Removed <i>Appendix E</i> . 7. Increased document margin. 8. Remove <i>Appendix D Type Definition</i> . Data Model Diagram and XML schema is

			<p>sufficient to define types.</p> <p>9. Added section 1.1 <i>Motivation</i>.</p> <p>10. Added <i>Appendix C.IV Sample DDL Document for Smart Oven</i>.</p>
1.1.1	July 3 2008	Chao Chen	<ol style="list-style-type: none"> <li>1. XML schema for DDL: DDL.xsd (Appendix B)</li> <li>2. Sample application of DDL sensor driver generator (Appendix E).</li> <li>3. Revision history table (Appendix F).</li> <li>4. Structural revision. <ol style="list-style-type: none"> <li>4.1. Remove “Data_type” element. The descriptions of data types are defined in the XML schema file DDL.xsd.</li> <li>4.2. Change “Id” element in Signal into an attribute.</li> <li>4.3. Disable non-numeric representation in “Range” element.</li> </ol> </li> </ol>
1.1.0	June 2008	Raja Bose	<ol style="list-style-type: none"> <li>1. Structural revision: <ol style="list-style-type: none"> <li>1.1. Merge “Id” and “Physical” element into Description.</li> <li>1.2. Add “Humidity” element for Operation_environment.</li> <li>1.3. Add run-time elements “UniqueId” and “Location”.</li> </ol> </li> </ol>

			<p>1.4. Add element “Device_type” to support virtual sensors.</p> <p>1.5. Change the name of the element “Description” to “Verbose_description.”</p> <p>2. Support for virtual sensors.</p> <p>3. Sample application documents for basic virtual sensors (Appendix B.II) and derived virtual sensors (Appendix B.III).</p> <p>4. Overview section.</p>
1.0.0	May 2008	Chao Chen	<p>1. The initial draft of the specification.</p>

## Table of Contents

1.	Overview .....	1
1.1.	Motivation .....	1
1.2.	What is a Device? .....	2
1.3.	Categories of Devices.....	3
1.4.	Device Operations.....	3
1.4.1.	Categories of Signals.....	3
1.4.2.	Sensor Operations .....	4
1.4.3.	Actuator Operations .....	6
1.4.4.	Complex Device Operations .....	6
1.4.4.1.	Safety Mechanisms for Devices .....	6
1.5.	DDL Applications.....	7
1.5.1.	A Sample Application of DDL Device Driver Generator.....	7
1.6.	DDL Structure .....	9
1.7.	Sample Document.....	9
2.	DDL Data Element Nomenclature:.....	9
3.	Framework Elements: .....	10
3.1.	<DDL> Element .....	10
3.2.	<Sensor> <Actuator> and <Device> Element.....	11
3.2.1.	<Description> Element.....	12
3.2.3.	<Interface> Element.....	14
4.	XML Considerations.....	17
4.1.	Namespace.....	17
4.2.	Schema .....	17
4.3.	Character Set.....	17

Appendix A: Data Model.....	18
Appendix B: XML Schema for DDL (DDL.xsd).....	21
Appendix C.I: Sample DDL Document for Physical Sensor .....	30
Appendix C.II: Sample DDL Document for Basic Virtual Sensor .....	34
Appendix C.III: Sample DDL Document for Derived Virtual Sensor .....	38
Appendix C.IV: Sample DDL Document for Smart Oven .....	38
Appendix C.V: Sample DDL Document for AnD Blood Pressure Monitor.....	49
Appendix D: References.....	554

## **1. Overview**

This document is a specification of the Device Description Language (DDL). DDL is an XML-based language which is being developed to initially support the integration of sensors to the intelligent environments. However, the specification is being written with enough flexibility to accommodate actuators and more complicated devices such as blood pressure monitors. DDL enables a uniform schema to describe sensors and devices and defines a proposal to the Service Oriented Device Architecture (SODA) standard.

### **1.1. Motivation**

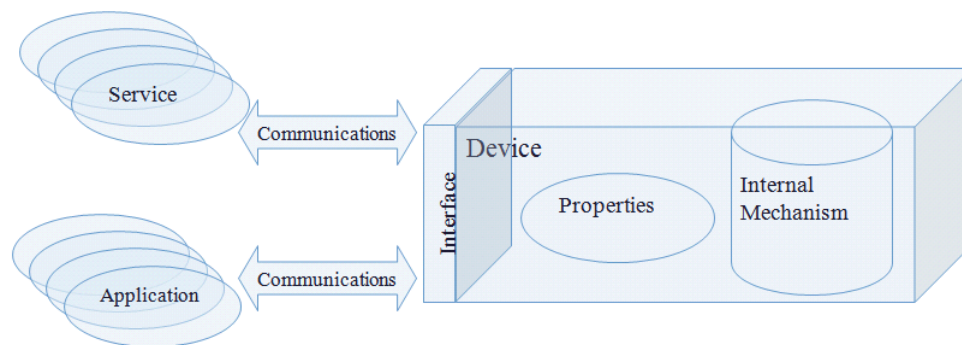
Devices are the building blocks of an intelligent environment. In a typical intelligent environment such as smart home, there is a wide variety of devices ranging from small pinhead sensors such as floor pressure sensors to large complex appliances such as an air conditioner. Even within the same category of devices, the interfaces and protocols used usually vary from manufacturer to manufacturer, from model to model. Therefore, it poses a serious challenge for programmers to connect and interface with all these devices poses an extra challenge to the programmers. Is there a neat solution that generalizes these devices so that the integration of them would be less ad hoc? DDL is one of the first attempts to employ a description language to define devices and their operations.

DDL is a proposal to the SODA standard, whose major goal of SODA is to represent devices such as sensors and actuators as services in today's enterprise SOA. With well-defined interfaces, software services are everything application developers need to know to bring together the devices and program the smart space. They should not worry about the nitty-gritty details of each ports and pins. However, the reality is that wiring and interfacing with devices of all types has been one of the major headaches of the real-world developer. This community needs a convenient solution that closes the gap between the physical devices and digital world. DDL provides a standard tool that describes a variety of devices in a common language that are

readable to both human users and computer programs. It creates a common interface for sensors and actuators and enables automatic integration of these devices in the programmable smart space.

## 1.2. What is a Device?

In DDL, a device is characterized as an entity consisting of a set of properties, some internal mechanism, and an interface (Figure 1). The properties provide information about a device such as its purpose, its capabilities, vendor and operating requirements. These are critical information for both system integrators and service programmers. The internal mechanisms are responsible for the operation of the device and unknown to the external world. The gap between the internal mechanism and the external world is bridged by the interface of the device. It specifies device IO and provides guidance to applications and other services to interact with the device. For example, the interface of a digital blood pressure monitor is responsible for parsing the byte streams to meaningful data that are sent through the serial port of the monitor.



**Figure 1. Characterizing a Device**

From a service-oriented perspective, the characteristics of a device which are essential to its utilization by an external user are its properties and interface. The properties provide information about a device such as its purpose, its capabilities, vendor and operating requirements. The



interface defines how a device interacts with its external users and provides ways to access the device either to get information from it and/or to control it.

### **1.3. Categories of Devices**

Devices can be classified into three categories: Sensor, Actuator and Complex Device.

- **Sensor:** A sensor is a device which only provides input to the external user.
- **Actuator:** An actuator is a device which only accepts output from the external user.
- **Complex Device:** A complex device is one which can both accept output from the external user and provide input to the external user.

### **1.4. Device Operations**

From a data-oriented perspective, a device can be modeled as a process consisting of inputs, data processing and output. In a service-oriented architecture, each device has a service representation which provides an interface to its properties and abstracts away its internal operation. Hence, in this document we describe the operation of devices in terms of input/processing/output from the perspective of their respective service entities. DDL describes the communications between a device and its corresponding service entity as ‘Signals’. Depending on the type of device, these communications can be unidirectional or bidirectional. In the temperature sensor example we present in Appendix C.I, we have unidirectional signals as the sensor can only push out data. ‘Readings’ are the meaningful information that the service entity receive from its associated device. It usually specifies a conversion or a parsing method that converts raw signals to real life data. For example, a reading of the temperature sensor (in unit of Centigrade) is converted from a 10-bit signal read from the ADC pin. In DDL, the combination of Signals and Readings together define the Interface element.

#### **1.4.1. Categories of Signals**

Signals can be categorized as: Input (signal flowing from member device to service object) or Output (signal flowing from service object to member device). Signals can also be categorized on the basis of their format as follows:

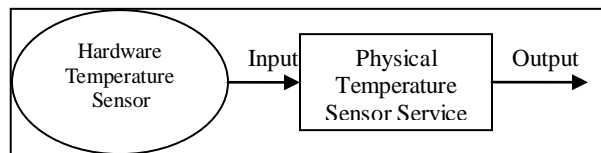
- **Analog:** This signal originates from the Analog-to-Digital Converter (ADC) of the connection hardware utilized by the device (such as a sensor platform). The range of values for this signal is  $[0, 10^n - 1]$  where 'n' is the resolution of the ADC in bits.
- **Digital:** This signal originates from a single pin of the connection hardware. The range of values for this signal is  $\{0, 1\}$ .
- **Protocol:** This signal originates from devices which have their own built-in communication protocol. This protocol may be standard (such as TCP/IP) or proprietary.

#### 1.4.2. Sensor Operations

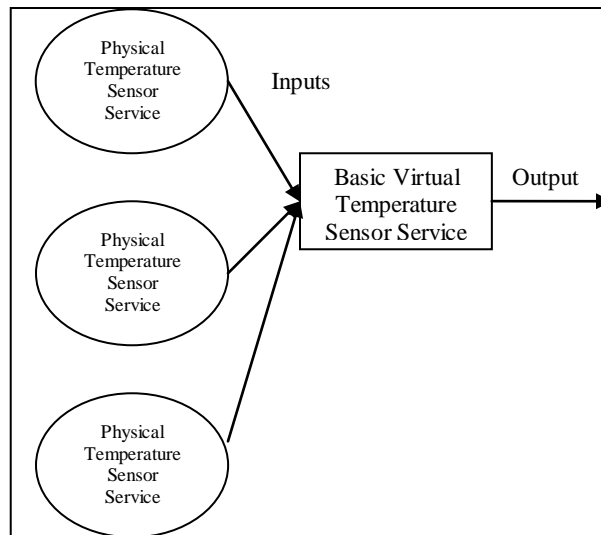
A sensor service can receive multiple inputs (signals) and provides a single output (reading) after processing them. Based on the number and type of inputs, we classify sensor services into three categories:

- **Physical (singleton) Sensor service:** A physical sensor service represents a hardware sensor. It is the most basic type of sensor service and is responsible for receiving input from the hardware sensor and processing the raw input into real world output (Figure 2(a)).
- **Basic Virtual Sensor service:** A Basic Virtual Sensor service is composed of a set of physical sensor services which detect the same type of phenomenon. Unlike the physical sensor service which is directly bound to a hardware sensor, a virtual sensor service represents a purely software sensor. A Basic Virtual Sensor service accepts multiple inputs from its member physical sensor services, processes them using aggregation techniques to produce a single output (Figure 2(b)). It is important to note that all the inputs to a Basic Virtual Sensor service are of the same type in terms of measurement units.

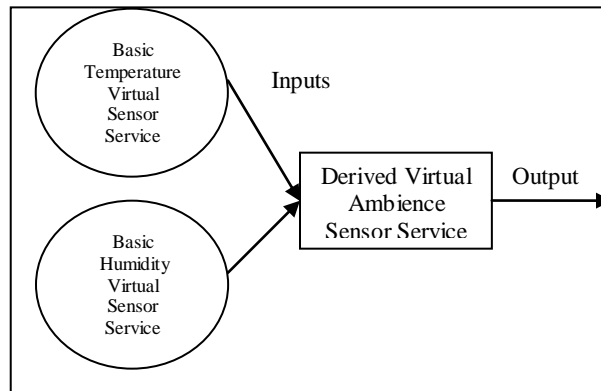
- Derived Virtual Sensor service:** A Derived Virtual Sensor service is composed of a set of Basic Virtual Sensor services, each of which detect a different type of phenomenon. As the name indicates, the output of this type of virtual sensor service is derived from multiple inputs. Hence, the type and measurement unit of output is totally different from that of any of the member sensor services (Figure 2(c)).



**Figure 2(a). Example of a Physical Sensor service**



**Figure 2(b). Example of a Basic Virtual Sensor service**



**Figure 2(c). Example of a Derived Virtual Sensor service**

### 1.4.3. Actuator Operations

An actuator service provides outputs to the hardware actuator device for controlling its operation. An external user utilized methods specified in the service interface to cause the transmission of output signals to the hardware device which then operates accordingly.

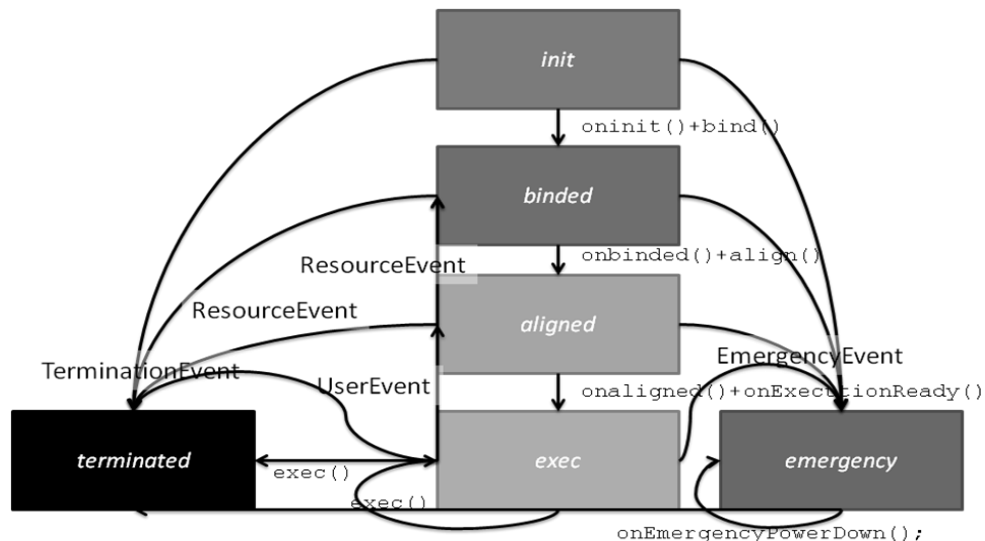
### 1.4.4. Complex Device Operations

A complex device service provides outputs to the hardware device it represents and also gathers input from it. Unlike sensor and actuators, communication between a complex device service and its corresponding hardware device is bi-directional.

#### 1.4.4.1. Safety Mechanisms for Devices

To smooth operations and enhance control and tracking of services in dynamic environment typical for pervasive computing, DDL adopts a device protection mechanism [6] which is based on a state machine for all devices as shown in Figure 3. *init* is the state when a device service object first instantiated; a device service is *binded* when successfully acquired all dependent services; before execution a service needs to be *aligned* with users' preferences and limitations; only then can a service start executing. When emergency hits, a device service can move into

emergencyPowerDown from any other state. Service safety interface is the manifestation of the transitions in the state machine. Allowed states of the device are specified in DDL for each operation.



**Figure 3. State Diagram for Devices**

## 1.5. DDL Language Processor

DDL is based on the concept that the availability of standard device descriptions allows for the development of intelligent environment applications integrating multiple sensors, actuators and complex devices. By representing each device as a service in the service-oriented architecture (SOA), programming the smart space becomes as easy as calling a number of methods from the device services. This subsection provides an example project that shows how DDL fits into the SOA framework and supports the programmable smart space.

### 1.5.1. A Sample Application of DDL Language Processor

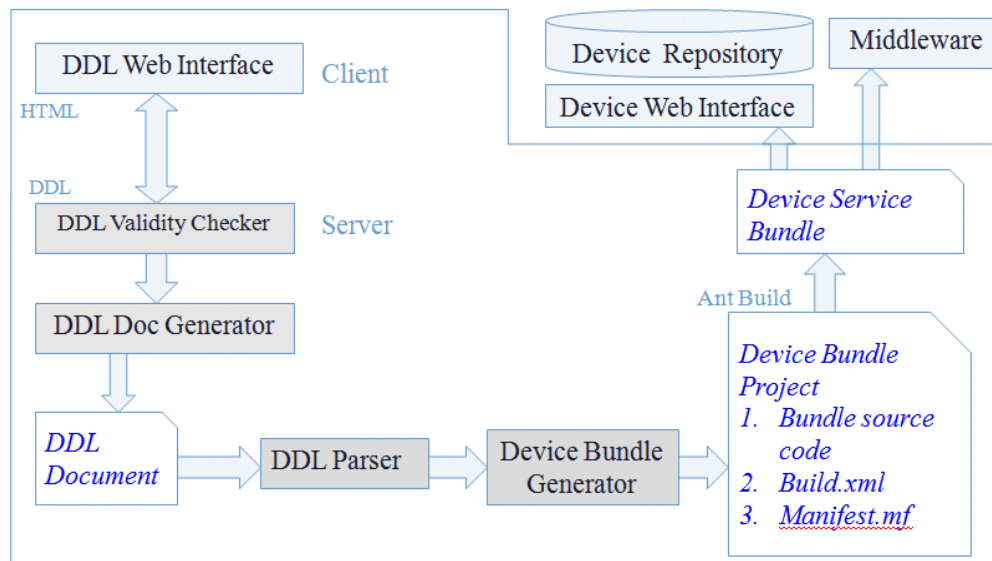
The example showcased in this section is an application that uses DDL language processor to automatically generate device service bundles (also termed as device drivers in the DDL language processor). In this example, the output of the application is a service bundle program, which is a software entity that represents the device in the SOA framework. It describes the functions and properties of the device and allows interfacing with the other services and applications in an Atlas-enabled software environment [5].

At the client side, user checks in the DDL document for a certain device through the DDL web interface. The DDL document can be provided by the device manufacturer or generated by the web script as user fills in the necessary information in the HTML form. A DDL validity checker runs at the server side and examines uploaded information for syntax errors. The errors and warning information will show up on the webpage if there is any. If the errors are cleared, the DDL document generator will create a downloadable DDL document for the device.

The DDL file device is then passed to the DDL parser and the service bundle generator. This software analyzes the DDL document and creates a bundle project for the device. The project includes the source code for the bundle program, build file and manifest information and is buildable using Apache Ant 1.7.0 or higher.

Once the project file is prepared, they will be built automatically into a device bundle and uploaded to the device bundle repository. The user or other programmers can use the device described by loading the device bundle and integrated it in their applications.

Figure 4 shows the process and data flow described above. The area in the box is the scope of the DDL Language Processor.



**Figure 4. Using DDL Language Processor to generate Service Bundles.**

## 1.6. DDL Structure

The tree diagrams in Appendix A provide a graphical representation of how DDL elements and attributes relate to each other. These relationships in concert with the DDL type definitions in Appendix C form the basis for language validation.

## 1.7. Sample Document

In addition to providing a definition of the elements and their attributes, this specification provides sample DDL documents in Appendix B for the Pressure Sensor proposed in the DDL Requirements document.

## 2. DDL Data Element Nomenclature:

This specification uses the following approach to describing elements:

- 1) Element name starts with upper case. Attribute names are all lower case.
- 2) Element and attribute names use a hyphen (“-”) to separate multiple word names so as to improve readability (ex. <creation-date>). Attribute and element names avoid the use of abbreviations to enhance readability.
- 3) Within this document, child elements are nested in a sub-paragraph under their parents.
- 4) Attributes are also nested but do not have the angle brackets (“<” and “>”) and are italicized.
- 5) The element’s and attribute’s type is provided in braces (“{ }”). For more information on each type refer to Appendix E.
- 6) If the element can occur zero or more times, an asterisk (“\*”) is placed after its name.
- 7) If the element occurs zero or one times, a question mark (“?”) follows its name.
- 8) If the element must appear at least once, a plus sign (“+”) trails its name.
- 9) Element names without a special trailing character must occur exactly once.
- 10) Each DDL specification references the requirement that it is designed to meet. The requirement is positioned at the end of the specification and contained in parentheses.
- 11) The order of element descriptions in the paragraphs below is not significant. Any required ordering of elements is specified in the tree diagrams found in Appendix A and type definitions in Appendix C.

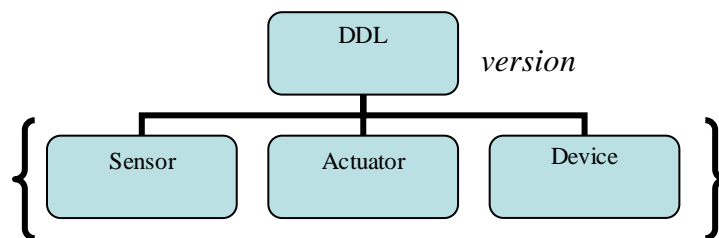
### **3. Framework Elements:**

#### **3.1. <DDL> Element**



The <DDL> element {sd:DDLType}[+]: The root element for DDL. It contains the following attributes and elements:

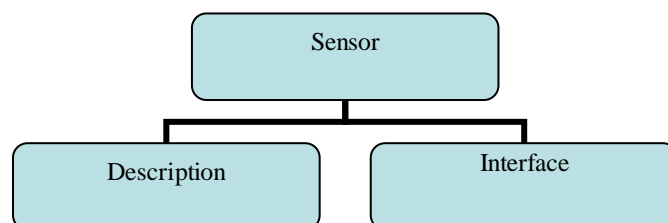
- a. Version attribute: The *version* attribute {xsd:string} indicates which version of DDL the instance contains.
- b. <Sensor>, <Actuator> or <Device> element: Depending on the type of device that the document is describing, one of these elements are used as the child of <DDL> elements.



### 3.2. <Sensor>, <Actuator> and <Device> Element

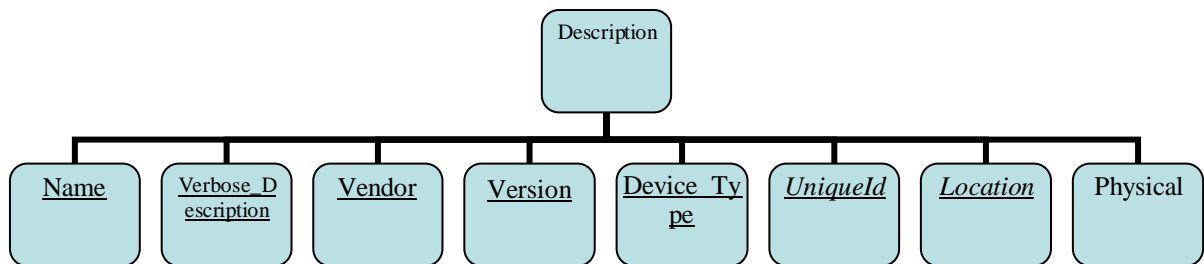
The <Sensor> element {sd:SensorType}[+] indicates the device described in the document is a sensor. DDL 1.2 also allows <Actuator> or <Device> at this level to replace <Sensor> if the type of device being described is actuator or other devices. As shown in the graph, the <Sensor> element has the following contents:

- a. <Description> element: The <Description> element contains descriptive information about a device.
- b. <Interface> element: The <Interface> element describes the interfaces of a device.



### 3.2.1. <Description> Element

The <Description> element {sd:DescriptionType}[+] contains the identification information of the device. It is the first child element of <Sensor>. As shown in the graph, the <Description> element has the following contents:

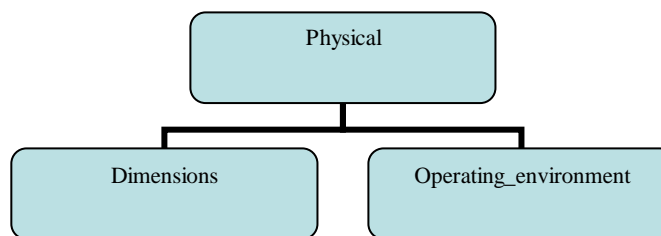


- a. <Name> element: The <Name> element {xsd:string}[+] gives the name of the device.
- b. <Verbose\_Description> element: The <Verbose\_Description> element {xsd:string} gives a text description of the device.
- c. <Vendor> element: The <Vendor> element {xsd:string} indicates the vendor of the device.
- d. <Version> element: The <Version> element {xsd:string} indicates the version of the device.
- e. <Device\_Type> element: The <Device\_Type> element {sd: DeviceTypeType} is the type of the device. It can be physical or virtual.
- f. <Physical> element: The <Physical> element {sd:PhysicalType} describes the physical characteristics of a device.
- g. <UniqueId> element: The <UniqueId> element {xsd:string} is a dynamic ID that is assigned to the device at run time.

- h. <Location> element: The <Location> element {xsd:string} is a dynamic value that is determined at run time.

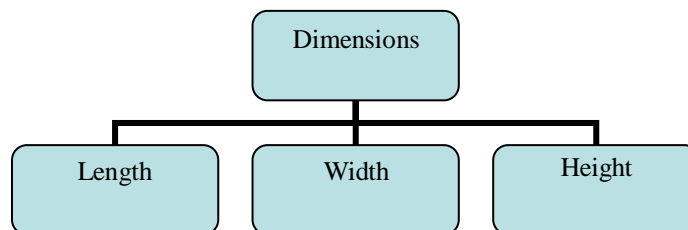
### 3.2.2. <Physical> Element

The <Physical> element {sd:PhysicalType} describes the physical characteristics of a device. It contains two elements: <Dimensions> and <Operating\_environment>.



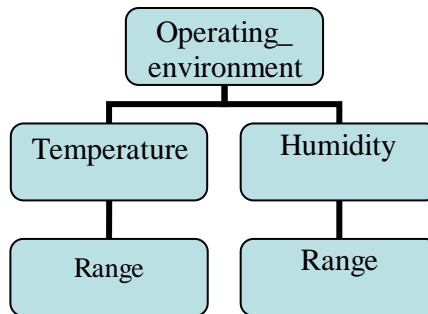
#### 3.2.2.1. <Dimensions> Element

The <Dimensions> element {sd:DimensionsType} describes the dimension of a device, including <Height>, <Width> and <Length>.



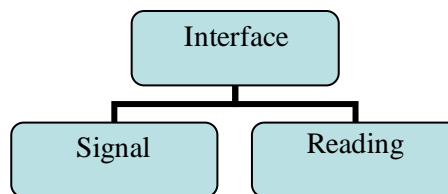
#### 3.2.2.2. <Operating\_environment> Element

The <Operating\_environment> element {sd: Operating\_environmentType} describes the permissible environmental parameters of a device, including <Temperature> and <Humidity>.



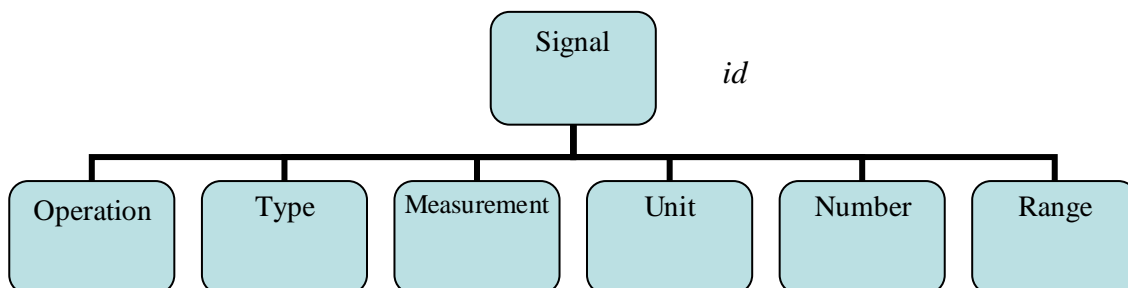
### 3.2.3. <Interface> Element

The <Interface> element {sd:InterfaceType}[+] contains the description of the interfaces of the device. It contains the following attributes and elements:



#### 3.2.3.1. <Signal> Element

The <Signal> element {sd:SignalType}[+] contains the description of signals transmitted via the interface. It contains the following attributes and elements:

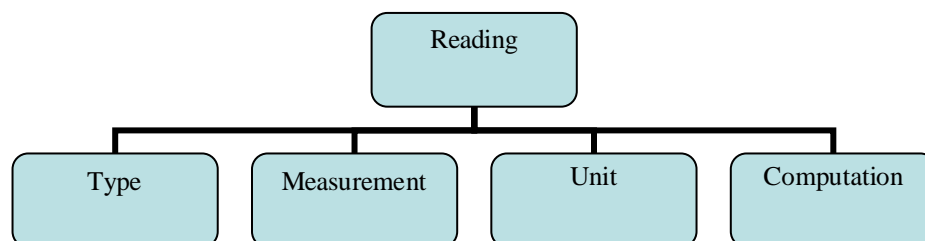


- a. *id* attribute: The *id* attribute{xsd: string}[+] is an exclusive alphabet-numeric value that is assigned to each signal.

- b. <Operation> element: The <Operation> element {sd: OperationType} specifies the types of operation. It can be either input or output.
- c. <Type> element: The <Type> element {sd: TypeType} describes the type of the signal. A Signal Type can be: Analog, Digital, Protocol or Logical. Analog/Digital is a low level collection of pins, Protocol is a high level interface to a device which has an in-built communication protocol (example: AnD Blood Pressure Monitor), Logical is high-level device service.
- d. <Measurement> element: The <Measurement> element {xsd: string} is a text explanation of the measurement of the signal. For example, it can be voltage, ADC value, or a byte sequences.
- e. <Unit> element: The <Unit> element {xsd: string} is a text explanation of the unit of the measurement.
- f. <Number> element: The <Number> element {sd: NumberType} defines whether it is a single signal or multiple signals of the same type.
- g. <Range> element: The <Range> element {sd: RangeType} describes the range of the signal value. It contains two elements: <Min> and <Max>.

### 3.2.3.2. <Reading> Element

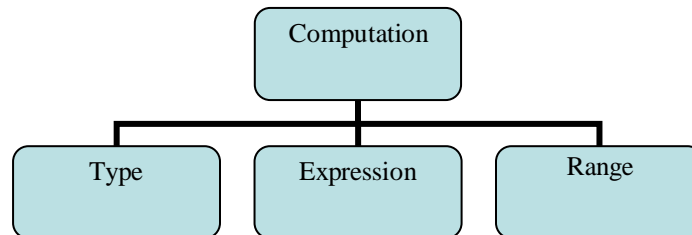
The <Reading> element {sd:ReadingType} contains information of readings converted from the signal. It carries the semantics of the signals. The element contains the following contents:



- a. <Type> element: The <Type> element {xsd: TypeType} is the type of readings from the device. It can be Basic for virtual sensors, Derived also for virtual sensors or Physical for singleton sensors.
- b. <Measurement> element: The <Measurement> element {xsd: string} is a text explanation of the measurement carried by the reading. For example, the measurement for a temperature sensor reading is temperature.
- c. <Unit> element: The <Unit> element {xsd: string} is a text explanation of the unit of the measurement. For example, the unit of reading in the above example can be Centigrade.
- d. <Computation> element: The <Computation> element {sd: ComputationType} defines the process of signal to reading conversion.

#### 3.2.3.2.1.1. < Computation > Element

The <Computation> element {sd: ComputationType} defines the process of signal to reading conversion. It consists of the following elements:



- a. <Type> element: The <Type> element {xsd: string} defines the type of the computation. It can be Aggregate, Formula or Map.
- b. <Expression> element: The <Expression> element {xsd: string} is a string representation of the conversion method.

- c. <Range> element: The <Range> element {sd: RangeType} defines the range of the computation results.

## **4. XML Considerations**

### **4.1. Namespace**

The namespace of DDL contains two parts: XML standard (version 1.1) namespace: <http://www.w3.org/TR/REC-xml-names/> and the namespace defined in Appendix E.

### **4.2. Schema**

DDL will use XML Schema to perform validity checking.

### **4.3. Character Set**

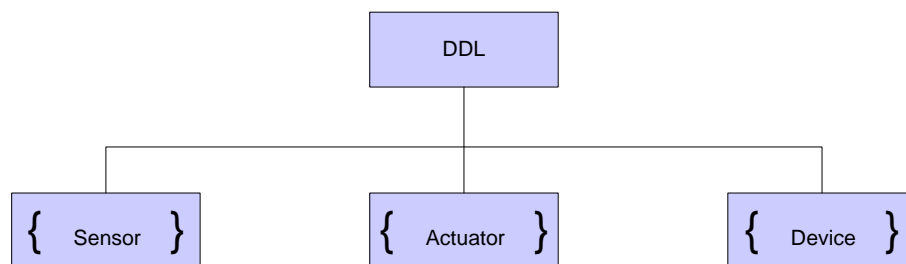
DDL will use UTF-8 encoding.

## Appendix A: Data Models

This appendix presents graph representations for data models of DDL devices. The tree graphs provide a hierarchical view of DDL data models of sensors, actuators and complex devices [Figure 1]. The graph is also a map for all the elements of the DDL language. In the map, each tree node represents an element whose contents are represented by its sub-tree. In addition, we use various symbols to represent the number of occurrences of each element [Table 1].

zero or one = ?	zero or more = *	one or more = +	choose one = { }
-----------------	------------------	-----------------	------------------

Appendix A. Table 1. Symbols notations for element occurrences.

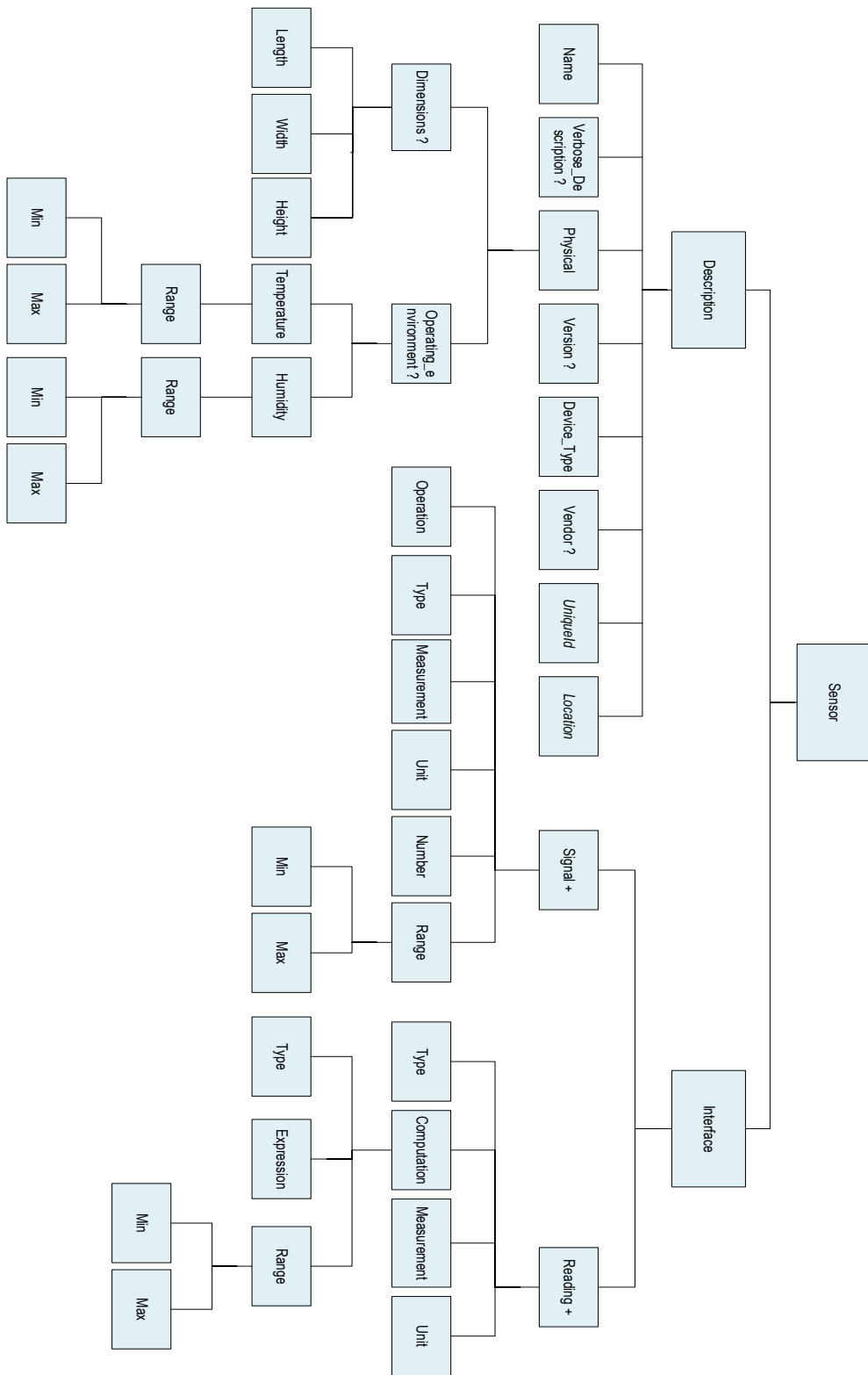


Appendix A. Figure 1. The data model of the DDL root element.

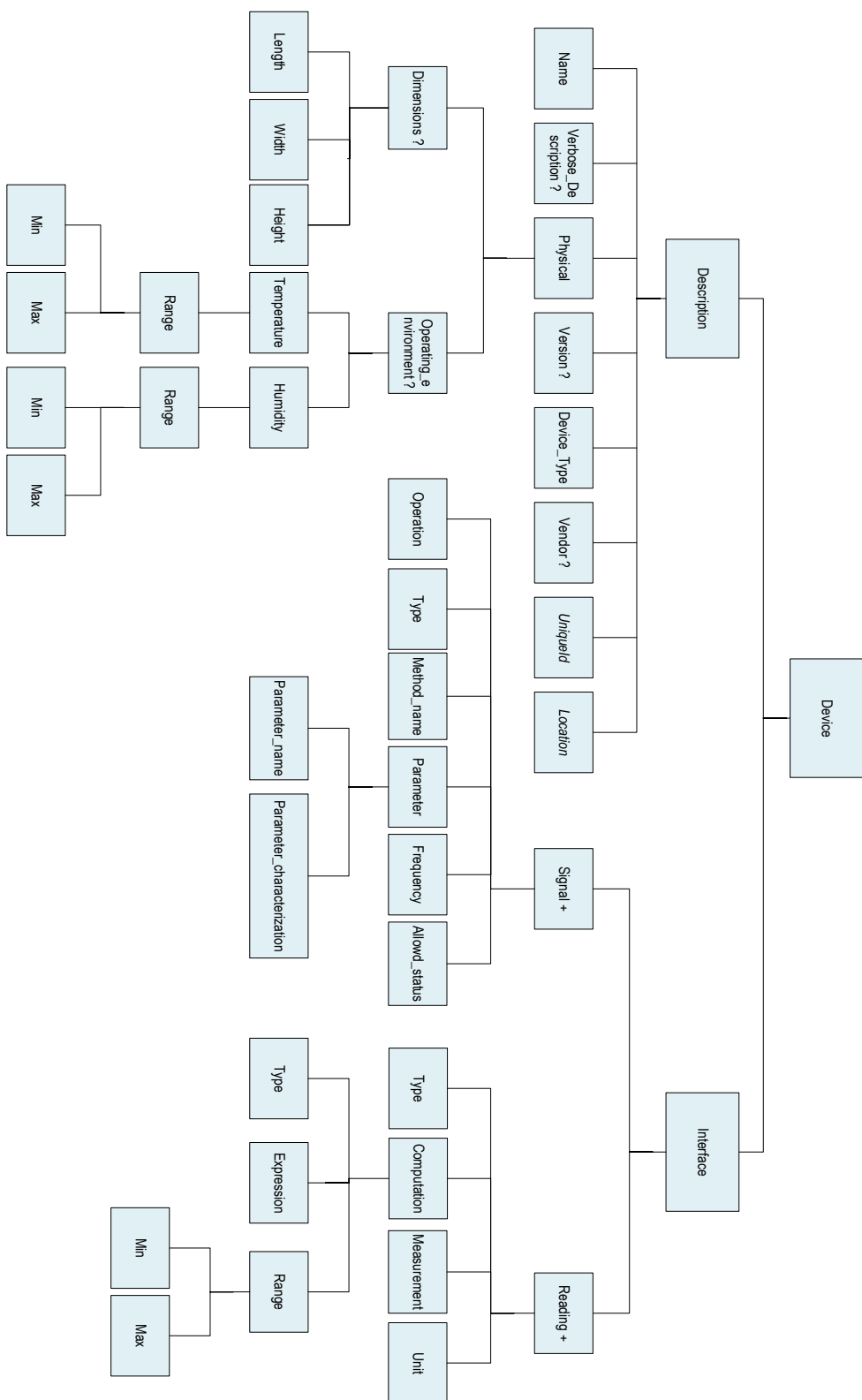
Figure 2 and Figure 3 in this appendix presents the hierarchical structure of data models for sensors and data models respectively.



Appendix A. Fig. 2:  
Data Models for Sensor



Appendix A, Fig. 3:  
Data Models for Complex Device



## Appendix B: XML Schema for DDL (DDL.xsd)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="DDL">
    <xsd:sequence>
      <xsd:element name="Sensor" type="Sensor" minOccurs="0" maxOccurs="1" />
      <xsd:element name="Actuator" type="Actuator" minOccurs="0" maxOccurs="1" />
      <xsd:element name="Device" type="Device" minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
    <xsd:attribute name="version" type="xsd:string" />
  </xsd:complexType>

  <xsd:complexType name="Sensor">
    <xsd:sequence>
      <xsd:element name="Description" type="Description" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Interface" type="Interface" minOccurs="1" maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Actuator">
    <xsd:sequence>
      <!-- Actuator Type is to be defined --> 
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Device">
    <xsd:sequence>
      <xsd:element name="Description" type="Description" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Interface" type="Interface" minOccurs="1" maxOccurs="1" /> 
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Description">
```

```

<xsd:sequence>
<xsd:element name="Name" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>
</xsd:element>
<xsd:element name="Device_Type" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="Physical" />
  <xsd:enumeration value="Virtual" />
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="Verbose_Description" minOccurs="0" maxOccurs="1">
<xsd:simpleType>
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>
</xsd:element>
<xsd:element name="Vendor" minOccurs="0" maxOccurs="1">
<xsd:simpleType>
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>
</xsd:element>
<xsd:element name="Version" minOccurs="0" maxOccurs="1">
<xsd:simpleType>
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>
</xsd:element>
<xsd:element name="Physical" type="Physical" minOccurs="1" maxOccurs="1" />
<xsd:element name="UniqueId" type="UniqueId" minOccurs="0" maxOccurs="1" />
<xsd:element name="Location" type="Location" minOccurs="0" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>

```

```

<xsd:complexType name="Physical">
<xsd:sequence>
  <xsd:element name="Dimensions" type="Dimensions" minOccurs="0" maxOccurs="1" />
  <xsd:element name="Operating_environment" type="Operating_environment"
    minOccurs="0" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Dimensions">
<xsd:sequence>
<xsd:element name="Length" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
  <xsd:restriction base="xsd:float" />
</xsd:simpleType>
</xsd:element>
<xsd:element name="Width" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
  <xsd:restriction base="xsd:float" />
</xsd:simpleType>
</xsd:element>
<xsd:element name="Height" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
  <xsd:restriction base="xsd:float" />
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Operating_environment">
<xsd:sequence>
  <xsd:element name="Temperature" type="Temperature" minOccurs="0" maxOccurs="1" />
  <xsd:element name="Humidity" type="Humidity" minOccurs="0" maxOccurs="1" />
</xsd:sequence>

```

```

</xsd:complexType>

<xsd:complexType name="Temperature">
<xsd:sequence>
  <xsd:element name="Range" type="Range" minOccurs="1" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Humidity">
<xsd:sequence>
  <xsd:element name="Range" type="Range" minOccurs="1" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Range">
<xsd:sequence>
<xsd:element name="Max" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
  <xsd:restriction base="xsd:float" />
</xsd:simpleType>
</xsd:element>
<xsd:element name="Min" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
  <xsd:restriction base="xsd:float" />
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="UniqueId">
  <xsd:sequence>
    <!-- Encoding scheme to be decided --> 
  </xsd:sequence>
  <xsd:attribute name="type" type="xsd:string" />

```

```

</xsd:complexType>

<xsd:complexType name="Location">
  <xsd:sequence>
    <!-- Encoding scheme to be decided -->
  </xsd:sequence>
  <xsd:attribute name="type" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="Interface">
<xsd:sequence>
  <xsd:element name="Signal" type="Signal" minOccurs="0" maxOccurs="unbounded" />
  <xsd:element name="Reading" type="Reading" minOccurs="0" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Signal">
<xsd:sequence>
<xsd:element name="Operation" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="Input" />
  <xsd:enumeration value="Output" />
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="Type" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="Analog" />
  <xsd:enumeration value="Digital" />
  <xsd:enumeration value="Protocol" />
  <xsd:enumeration value="Logical" />
</xsd:restriction>

```

```

    </xsd:simpleType>
  </xsd:element>
<xsd:element name="Measurement" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
</xsd:element>
<xsd:element name="Unit" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string" />
  </xsd:simpleType>
</xsd:element>
<xsd:element name="Number" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Multiple" />
      <xsd:enumeration value="Single" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
<xsd:element name="Range" type="Range" minOccurs="0" maxOccurs="1" />
<xsd:element name="Method_name" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
<xsd:element name="Parameter" type="Parameter" minOccurs="0" maxOccurs="unbounded"
  />
<xsd:element name="Frequency" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
    </xsd:restriction>
  </xsd:simpleType>

```



```

</xsd:element>
<xsd:element name="Allowed_Status" minOccurs="0" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="INIT " />
  <xsd:enumeration value="BINDED" />
  <xsd:enumeration value="ALIGNED" />
  <xsd:enumeration value="EXEC" />
  <xsd:enumeration value="TERMINATE" />
  <xsd:enumeration value="EMERGENCY" />
  <xsd:enumeration value="ERROR" />
  <xsd:enumeration value="ALL" />
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="id" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="Parameter">
<xsd:sequence>
<xsd:element name="Parameter_name" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
  </xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="Parameter_characterization" type="xsd:float" minOccurs="0"
  maxOccurs="unbounded"/>
</xsd:sequence>
<xsd:attribute name="id" type="xsd:string" use="required" />
<xsd:attribute name="enumeration" type="xsd:boolean" use="required" />
<xsd:attribute name="discrete" type="xsd:boolean" use="required" />
</xsd:complexType>

```

```

<xsd:complexType name="Reading">
  <xsd:sequence>
    <xsd:element name="Type" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="Basic" />
          <xsd:enumeration value="Derived" />
          <xsd:enumeration value="Physical" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="Measurement" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string" />
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="Unit" minOccurs="0" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string" />
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="Computation" type="Computation" minOccurs="0" maxOccurs="1" />
    <xsd:element name="Range" type="Range" minOccurs="0" maxOccurs="1" />
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="Computation">
  <xsd:sequence>
    <xsd:element name="Type" minOccurs="1" maxOccurs="1">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">

```

```
<xsd:enumeration value="Aggregate" />
<xsd:enumeration value="Formula" />
<xsd:enumeration value="Map" />
  </xsd:restriction>
  </xsd:simpleType>
  </xsd:element>
<xsd:element name="Expression" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string" />
    </xsd:simpleType>
  </xsd:element>
  </xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

## Appendix C.I: Sample DDL Document for Physical Sensor

```
<?xml version="1.0" encoding="UTF-8" ?>
<DDL version="1.21">
  <Sensor>
    <!-- General information of the device -->
    <Description>
      <!-- Name of the device -->
      <Name>Temperature Sensor </Name>
      <!-- Type of Device (Physical(Singleton) or Virtual) -->
      <Device_Type>Physical</Device_Type>
      <!-- Description of the device -->
      <Verbose_Description>
TMP36 Analog Temperature Sensor
      </Verbose_Description>
      <!-- Device vendor -->
      <Vendor>University of Florida </Vendor>
      <!-- Device version -->
      <Version>1.0 </Version>
      <Physical>
        <!-- Dimensions of the device -->
        <Dimensions>
          <!-- Length in mm, left blank if unknown -->
          <Length>24</Length>
          <!-- Width in mm, left blank if unknown -->
          <Width>34</Width>
          <!-- Height in mm, left blank if unknown -->
          <Height>24</Height>
        </Dimensions>
        <!-- Permissible environment for operation -->
        <Operating_environment>
          <!-- Permissible temperature range for operation -->
          <Temperature>
            <Range>
```

```

        <!-- left blank if unknown -->
        <Min></Min>
        <!-- left blank if unknown -->
        <Max></Max>
    </Range>
</Temperature>
<!-- Permissible humidity range for operation -->
<Humidity>
    <Range>
        <!-- left blank if unknown -->
        <Min></Min>
        <!-- left blank if unknown -->
        <Max></Max>
    </Range>
</Humidity>
</Operating_environment>
</Physical>
</Description>
<!--Interfaces of a device-->
<Interface>
    <!-- To avoid confusion, ensure Signal id is always alpha-numeric instead of numeric -->
    <Signal id="s1">
        <!-- Value of Operation attribute can be Input or Output -->
        <Operation>Input</Operation>
        <!-- A Signal Type can be: Analog or Digital or Protocol or Logical
        Analog/Digital is a low level collection of pins
        Protocol is a high level interface to a device which has an
        in-built communication protocol (example: AnD Blood Pressure Monitor)
        Logical is high-level device service.
        -->
        <Type>Analog</Type>
        <!-- Value of Measurement attribute can be: ADC, Digital or a string whose value
        is equal to the Reading->Measurement attribute of another physical/virtual sensor
        -->
        <Measurement>ADC</Measurement>
    </Signal id="s1">

```







```

<Unit>null</Unit>
<!-- Number can be Single or Multiple (many signal inputs of same type) -->
<Number>Single</Number>
<Range>
    <!-- leave blank if you don't know -->
    <Min>0</Min>
    <!-- leave blank if you don't know -->
    <Max>1023</Max>
</Range>
</Signal>
<Reading>
    <!-- Type can be: Basic(Virtual Sensor), Derived(Virtual Sensor),
         Physical (Singleton Sensor)
    -->
    <Type>Physical</Type>
    <Measurement>Temperature</Measurement>
    <Unit>Centigrade</Unit>
    <Computation>
        <!-- Possible value of the Type attribute can be: Aggregate, Formula or Map -->
        <Type>Formula</Type>
        <!-- Possible valid Expression attribute values can be:
            For Type='Aggregate': Mean, Median, Mode, Max, Min, Sum
            For Type='Formula': <numerical expression as function of signal ids>
            For Type='Map': <map of signal ids to range of output values>
        -->
        <!-- The formula function conforms to Java syntax -->
        <Expression>
            reading = (((s1/1023)*3.3)-0.5)*(1000/10)
        </Expression>
        <!--Range can be calculated automatically -->
        <Range>
            <Min></Min>
            <Max></Max>
        </Range>
    </Computation>

```

```
</Reading>  
</Interface>  
</Sensor>  
</ DDL >
```

## Appendix C.II: Sample DDL Document for Basic Virtual Sensor

```
<?xml version="1.0" encoding="UTF-8" ?>
<DDL version="1.21">
  <Sensor>
    <!-- General information of the device -->
    <Description>
      <!-- Name of the device -->
       <Name>  Pressure Sensor </Name>
      <!-- Type of Device (Physical(Singleton) or Virtual) -->
      <Device_Type>Virtual</Device_Type>
      <!-- Description of the device --> 
      <Verbose_Description>Atmospheric Pressure Sensor</Verbose_Description>
      <!-- Device vendor -->
      <Vendor>  University of Florida </Vendor>
      <!-- Device version -->
      <Version>1.0 </Version>
      <Physical>
        <!-- Dimensions of the device -->
        <Dimensions>
          <!-- Length in mm (not available for virtual sensor) -->
          <Length>-1  </Length>
          <!-- Width in mm (not available for virtual sensor) -->
          <Width>-1  </Width>
          <!-- Height in mm (not available for virtual sensor) -->
          <Height>-1  </Height>
        </Dimensions>
        <!-- Permissible environment for operation -->
        <Operating_environment>
          <Temperature>
            <Range>
              <!-- Values of Runtime attributes are not mentioned -->
              <Min></Min>
            </Range>
          </Temperature>
        </Operating_environment>
      </Physical>
    </Description>
  </Sensor>
</DDL>
```



```


        <!-- Values of Runtime attributes are not mentioned -->
        <Max></Max>
    </Range>
</Temperature>
<Humidity>
    <Range>
        <!-- Values of Runtime attributes are not mentioned -->
        <Min></Min>
        <!-- Values of Runtime attributes are not mentioned -->
        <Max></Max>
    </Range>
</Humidity>
</Operating_environment>
</Physical>
<!-- Values of Runtime attributes are not mentioned only data type is provided -->
<UniqueId type=" alphanumeric"></UniqueId>
<!-- Values of Runtime attributes are not mentioned only data type is provided -->
<Location type=" vector"></Location>
</Description>
<!--Interfaces of a device-->
<Interface>
    <!-- To avoid confusion, ensure Signal id is always alpha-numeric instead of numeric -->
    <Signal id="s1">
        <!-- Value of Operation attribute can be Input or Output -->
        <Operation>Input</Operation>
        <!-- A Signal Type can be: Analog or Digital or Protocol or Logical
        Analog/Digital is a low level collection of pins
        Protocol is a high level interface to a device which has an
        in-built communication protocol (example: AnD Blood Pressure Monitor)
        Logical is high-level device service.
        -->
        <Type>Logical</Type>
        <!-- Value of Measurement attribute can be: Analog, Digital or a string
        whose value is equal to the Reading - > Measurement attribute of
        another physical/virtual sensor

```

```

-->
<Measurement>Pressure</Measurement>
<Unit>Pascal</Unit>
<!-- Number can be Single or Multiple (many signal inputs of same type) -->
<Number> Multiple</Number>
<Range>
  <!-- leave blank if you don't know -->
  <Min></Min>
  <!-- leave blank if you don't know -->
  <Max></Max>
</Range>
</Signal>
<Reading>
  <!-- Type can be: Basic(Virtual Sensor), Derived(Virtual Sensor),
        Physical (Singleton Sensor)
-->
  <Type> Basic </Type>
  <Measurement> Pressure </Measurement>
  <Unit>Pascal</Unit>
  <Computation>
    <!-- Possible value of the Type attribute can be: Aggregate, Formula or Map -->
    <Type>Aggregate</Type>
    <!-- Possible valid Expression attribute values can be:
        For Type='Aggregate': Mean, Median, Mode, Max, Min, Sum
        For Type='Formula': <numerical expression as function of signal ids>
        For Type='Map': <map of signal ids to range of output values>
-->
    <!-- The formula function conforms to Java syntax -->
    <Expression>
      Mean
    </Expression>
    <!--Range can be calculated automatically -->
    <Range>
      <Min></Min>
      <Max></Max>

```

```
        </Range>
      </Computation>
    </Reading> 
  </Interface>
</Sensor>
</DDL>
```

## Appendix C.III: Sample DDL Document for Derived Virtual Sensor

```
<?xml version="1.0" encoding="UTF-8" ?>
<DDL version="1.21">
  <Sensor>
    <!-- General information of the device -->
    <Description>
      <!-- Name of the device -->
      <Name>Ambience Sensor</Name>
      <!-- Type of Device (Physical(Singleton) or Virtual) -->
      <Device_Type> Virtual </Device_Type>
      <!-- Description of the device -->
      <Verbose_Description>
        Sensor for determining Ambience of room
      </Verbose_Description>
      <!-- Device vendor -->
      <Vendor>University of Florida </Vendor>
      <!-- Device version -->
      <Version>1.0 </Version>
      <Physical>
        <!-- Dimensions of the device -->
        <Dimensions>
          <!-- Length in mm, left blank if unknown -->
          <Length>-1</Length>
          <!-- Width in mm, left blank if unknown -->
          <Width>-1</Width>
          <!-- Height in mm, left blank if unknown -->
          <Height>-1</Height>
        </Dimensions>
        <!-- Permissible environment for operation -->

```

```

<Operating_environment>
  <!-- Permissible temperature range for operation -->
  <Temperature>
    <Range>
      <!-- left blank if unknown -->
      <Min></Min>
      <!-- left blank if unknown -->
      <Max></Max>
    </Range>
  </Temperature>
  <!-- Permissible humidity range for operation -->
  <Humidity>
    <Range>
      <!-- left blank if unknown -->
      <Min></Min>
      <!-- left blank if unknown -->
      <Max></Max>
    </Range>
  </Humidity>
</Operating_environment>
</Physical>
<!-- Values of Runtime attributes are not mentioned -->
<UniqueId type=" alphanumeric"></UniqueId>
<!-- Values of Runtime attributes are not mentioned -->
<Location type=" vector"></Location>
</Description>
<!--Interfaces of a device-->
<Interface>
  <!-- To avoid confusion, ensure Signal id is always alpha-numeric instead of numeric -->
  <Signal id="s1">
    <!-- Value of Operation attribute can be Input or Output -->

```

```

<Operation>Input</Operation>
<!-- A Signal Type can be: Analog or Digital or Protocol or Logical
      Analog/Digital is a low level collection of pins
      Protocol is a high level interface to a device which has an
      in-built communication protocol (example: AnD Blood Pressure Monitor)
      Logical is high-level device service.
-->
<Type>Logical</Type>
<!-- Value of Measurement attribute can be: ADC, Digital or a string whose value
      is equal to the Reading->Measurement attribute of another physical/virtual sensor
-->
<Measurement>Temperature</Measurement>
<Unit>Centigrade</Unit>
<!-- Number can be Single or Multiple (many signal inputs of same type) -->
<Number>Single</Number>
<Range>
  <!-- leave blank if you don't know -->
  <Min></Min>
  <!-- leave blank if you don't know -->
  <Max></Max>
</Range>
</Signal>
<Signal id="s2">
  <!-- Value of Operation attribute can be Input or Output -->
  <Operation>Input</Operation>
  <!-- A Signal Type can be: Analog or Digital or Protocol or Logical
        Analog/Digital is a low level collection of pins
        Protocol is a high level interface to a device which has an
        in-built communication protocol (example: AnD Blood Pressure Monitor)
        Logical is high-level device service.
  -->

```

```

<Type>Logical</Type>
<!-- Value of Measurement attribute can be: ADC, Digital or a string whose value
      is equal to the Reading->Measurement attribute of another physical/virtual sensor
-->
<Measurement>Humidity</Measurement>
<Unit>Percent</Unit>
<!-- Number can be Single or Multiple (many signal inputs of same type) -->
<Number>Single</Number>
<Range>
  <!-- leave blank if you don't know -->
  <Min></Min>
  <!-- leave blank if you don't know -->
  <Max></Max>
</Range>
</Signal>
<Reading>
  <!-- Type can be: Basic(Virtual Sensor), Derived(Virtual Sensor),
        Physical (Singleton Sensor)
-->
  <Type>Derived</Type>
  <!-- Ambience is a binary index indicating whether the environment is agreeable -->
  <Measurement>Ambience</Measurement>
  <Unit>Binary</Unit>
  <Computation>
    <!-- Possible value of the Type attribute can be: Aggregate, Formula or Map -->
    <Type>Map</Type>
    <!-- Possible valid Expression attribute values can be:
          For Type='Aggregate': Mean, Median, Mode, Max, Min, Sum
          For Type='Formula': <numerical expression as function of signal ids>
          For Type='Map': <map of signal ids to range of output values>
    -->

```

```

<!-- The formula function conforms to Java syntax -->
<Expression>
{s1=[18:27] && s2=[40:60] = 1, s1!= [18:27] || s2!= [40:60] = 0}
</Expression>
<!--Range can be calculated automatically -->
<Range>
    <Min></Min>
    <Max></Max>
</Range>
</Computation>
</Reading>
</Interface>
</Sensor>
</ DDL >

```



## Appendix C.IV: Sample DDL Document for Smart Oven




```
<?xml version="1.0" encoding="UTF-8" ?>
<DDL version="1.21">
  <Device>
    <!-- General information of the device -->
    <Description>
      <!-- Name of the device -->
      <Name>Smart Oven</Name>
      <!-- Type of Device (Physical(Singleton) or Virtual) -->
      <Device_Type> Physical </Device_Type>
      <!-- Description of the device -->
      <Verbose_Description>
        A conceptual smart oven for demonstration purposes
      </Verbose_Description>
      <!-- Device vendor -->
      <Vendor>University of Florida </Vendor>
      <!-- Device version -->
      <Version>1.0 </Version>
      <Physical>
        <!-- Dimensions of the device -->
        <Dimensions>
          <!-- Length in mm, left blank if unknown -->
          <Length></Length>
          <!-- Width in mm, left blank if unknown -->
          <Width></Width>
          <!-- Height in mm, left blank if unknown -->
          <Height></Height>
        </Dimensions>
        <!-- Permissible environment for operation -->
```

```

<Operating_environment>
  <!-- Permissible temperature range for operation -->
  <Temperature>
    <Range>
      <!-- left blank if unknown -->
      <Min></Min>
      <!-- left blank if unknown -->
      <Max></Max>
    </Range>
  </Temperature>
  <!-- Permissible humidity range for operation -->
  <Humidity>
    <Range>
      <!-- left blank if unknown -->
      <Min></Min>
      <!-- left blank if unknown -->
      <Max></Max>
    </Range>
  </Humidity>
</Operating_environment>
</Physical>
<!-- Values of Runtime attributes are not mentioned -->
<UniqueId type=" alphanumeric"></UniqueId>
<!-- Values of Runtime attributes are not mentioned -->
<Location type=" vector"></Location>
</Description>
<!--Interfaces of a device-->
<Interface>
  <!-- To avoid confusion, ensure Signal id is always alpha-numeric instead of numeric -->
  <Signal id="s1">
    <!-- Value of Operation attribute can be Input or Output -->

```

```

<Operation>Output</Operation>
<!-- A Signal Type can be: Analog or Digital or Protocol or Logical
      Analog/Digital is a low level collection of pins
      Protocol is a high level interface to a device which has an
      in-built communication protocol (example: AnD Blood Pressure Monitor)
      Logical is high-level device service.
-->
<!-- An output signal of type "protocol" is the high level commands that the device will
      receive. Such signal is in the format of "Method_name<Parameters>". Each signal is a
      command that consists of method name and a list of parameters.
-->
<Type>Protocol</Type>
<!-- Value of Measurement attribute can be: ADC, Digital or a string whose value
      is equal to the Reading->Measurement attribute of another physical/virtual sensor
-->
<Method_name>turnOn</Method_name>
<!-- enumeration and discrete are Boolean values that specify the range and
      resolution of the parameter -->
<Parameter id = "p1" enumeration="false" discrete="true">
  <!-- The verbose description of the parameter. Leave blank if unknown -->
  <Parameter_name>time</Parameter_name>
  <!-- for discrete range, first param is upper bound --> 
  <Parameter_characterization>100</Parameter_characterization>
  <!-- for discrete range, second param is upper bound --> 
  <Parameter_characterization>150</Parameter_characterization>
  <!-- for discrete range, third param is resolution --> 
  <Parameter_characterization>1</Parameter_characterization>
</Parameter>
<!--maximum allowed frequency of the signal (per second) -->
<Frequency>2</Frequency>




```

```

<!-- Overfrequent handler on -->
<Overfrequent_handler>f_on</Overfrequent_handler>
<!-- Allowed Status -->
<Allowed_status>INIT</Allowed_status>
</Signal>
<Signal id="s2">
  <!-- Value of Operation attribute can be Input or Output -->
  <Operation>Output</Operation>
  <!-- A Signal Type can be: Analog or Digital or Protocol or Logical
    Analog/Digital is a low level collection of pins
    Protocol is a high level interface to a device which has an
    in-built communication protocol (example: AnD Blood Pressure Monitor)
    Logical is high-level device service.
  -->
  <!-- An output signal of type "protocol" is the high level commands that the device will
    receive. Such signal is in the format of "Method_name<Parameters>". Each signal is a
    command that consists of method name and a list of parameters.
  -->
  <Type>Protocol</Type>
  <!-- Value of Measurement attribute can be: ADC, Digital or a string whose value
    is equal to the Reading->Measurement attribute of another physical/virtual sensor
  -->
  <Method_name>turnOff</Method_name>
  <!-- Leave parameter list blank if there is not any parameters -->
  <!--maximum allowed frequency of the signal (per second) -->
  <Frequency>2</Frequency>
  <!-- Overfrequent handler on -->
  <Overfrequent_handler>f_on</Overfrequent_handler>
  <!-- Allowed Status -->
  <Allowed_status>ALL</Allowed_status>
</Signal>

```

```

<Signal id="s3">
  <!-- Value of Operation attribute can be Input or Output -->
  <Operation>Output</Operation>
  <!-- A Signal Type can be: Analog or Digital or Protocol or Logical
  Analog/Digital is a low level collection of pins
  Protocol is a high level interface to a device which has an
  in-built communication protocol (example: AnD Blood Pressure Monitor)
  Logical is high-level device service.
  -->
  <!-- An output signal of type "protocol" is the high level commands that the device will
  receive. Such signal is in the format of "Method_name<Parameters>". Each signal is a
  command that consists of method name and a list of parameters.
  -->
  <Type>Protocol</Type>
  <!-- Value of Measurement attribute can be: ADC, Digital or a string whose value
  is equal to the Reading->Measurement attribute of another physical/virtual sensor
  -->
  <Method_name>setTemperature</Method_name>
  <!-- enumeration and discrete are Boolean values that specify the range and
  resolution of the parameter -->
  <Parameter id="p1" enumeration="false" discrete="true">
    <!-- The verbose description of the parameter. Leave blank if unknown -->
    <Parameter_name>temperature</Parameter_name>
    <!-- for discrete range, first param is upper bound --> 
    <Parameter_characterization>100</Parameter_characterization>
    <!-- for discrete range, second param is upper bound --> 
    <Parameter_characterization>150</Parameter_characterization>
    <!-- for discrete range, third param is resolution --> 
    <Parameter_characterization>4</Parameter_characterization>
  </Parameter>

```

```

        <!--maximum allowed frequency of the signal (per second) -->
        <Frequency>2</Frequency>
        <-- Overfrequent handler on -->
        <Overfrequent_handler>f_on</Overfrequent_handler>
        <!-- Allowed Status -->
        <Allowed_status>EXEC</Allowed_status>
    </Signal>
    <-- The smart oven is an appliance that only receives inputs. It does not have readings
        as it doesn't output data. The Reading element is left empty.
    -->
    <Reading>
    </Reading>
</Interface>
</Device>
</ DDL >

```

## Appendix C.V: Sample DDL Document for Blood Pressure Monitor

In this appendix, we present a sample DDL document for a commercially available digital device (AnD UA-767PC blood pressure monitor [Figure 1]). The device measures blood pressure (systolic and diastolic pressure) and heart rate and uses its internal memory to store measurement results. It has a serial port which allows bi-directional transmissions of data and commands. The sample DDL document shown in this appendix provides an example of modeling transmission protocols and characterizing device behaviors.



Figure 1. AnD UA-767PC Blood Pressure Monitor.

The blood pressure monitor has five different states:

- Stand\_by: The device operates in low power stage and shows the hour and minute of current time.
- Communication: This state starts when the device receives RS-232C commands from an external host through its serial port.
- Measurement: Under this mode, the cuff will start inflating.

- Display: It is the time to show the measurement result on the LCD display.
- Time\_setting: User can set the time and date into the device.
- Memory\_display: The device displays previous measurement results.

Figure 2 is a state diagram of the blood pressure monitor and shows how the device moves among these states. RS-232C commands that trigger state transitions are marked on the arrows between states. The format and descriptions of these commands are listed in Table 1:

Table 1. The RS-232C Command Set for UA-767PC

Command Name	ASCII Code	Descriptions
End of transmission	30 + 34	End of transmission (Off line request)
Open port	30 + 35	Open communication port (On line request)
Start measuring	34 + 30	Start blood pressure measuring
Set time & date	33 + 31	Set time & date to the device
Inquire data from mem.	31 + 30	Inquire blood pressure & pulse data from memory
Time out*	NA	Time out is not a command. The device resumes stand_by mode after a period of time without receiving any commands.

In the sample document provided in this appendix, we will show:

1. How to use DDL to represent the finite state machine that describes the behavior of the device.
2. How to use DDL to represent the data format of the protocols and their semantics.



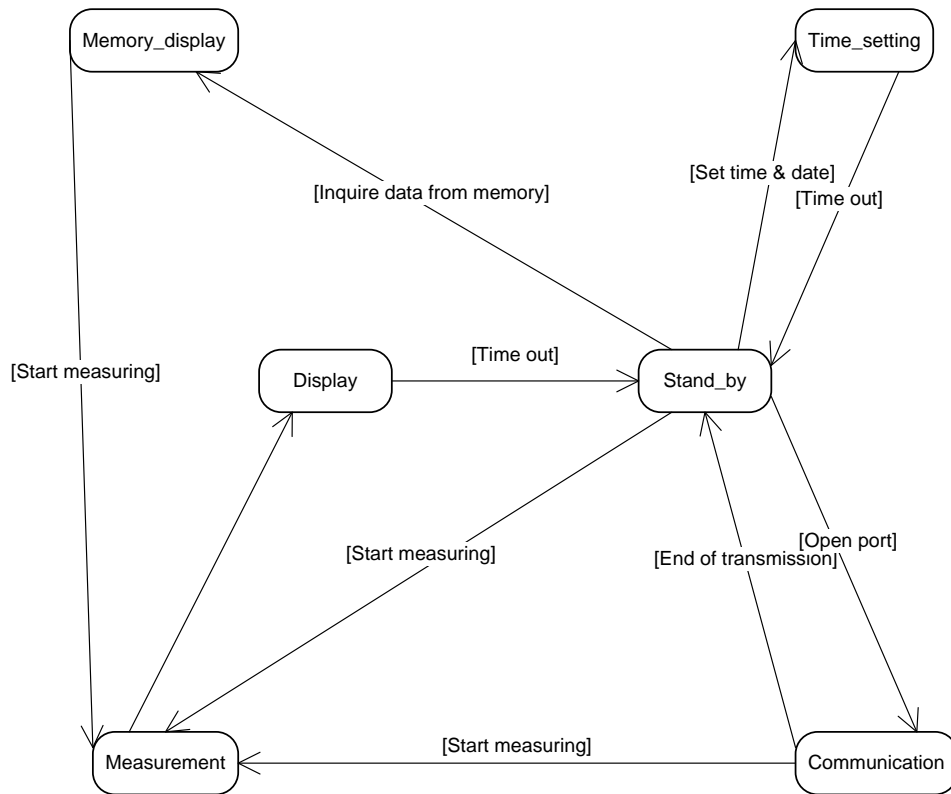


Figure 2. The State Diagram of UA-767PC

The sample document for the blood pressure monitor is shown below:

```

<DDL version="1.3">
  <Device>
    <Description>
      <Name>AnDBPMonitor</Name>
      <Device_Type>Physical</Device_Type>
      <Verbose_Description>
        AnD767PC Blood Pressure Monitor
      </Verbose_Description>
      <Vendor>University of Florida</Vendor>
      <Version>1.0</Version>
      <Physical>
        <Dimensions>

```

```

        <Length />
        <Width />
        <Height />
    </Dimensions>
    <Operating_environment>
        <Temperature>
            <Range>
                <Min />
                <Max />
            </Range>
        </Temperature>
        <Humidity>
            <Range>
                <Min />
                <Max />
            </Range>
        </Humidity>
    </Operating_environment>
</Physical>
<UniqueId type="alphanumeric" />
<Location type="vector" />
</Description>
<Interface>
    <Signal id="s1">
        <Operation>Input</Operation>
        <Type>Protocol</Type>
        <Measurement>Digital</Measurement>
        <Unit />
        <Number>Single</Number>
        <Range>
            <Min />
            <Max />
        </Range>
    </Signal>
    <Reading id="r1">
        <Type>Physical</Type>
        <Measurement>SysDiaDiff</Measurement>
        <Unit>mmHg</Unit>
        <Computation>
            <Type>Formula</Type>
            <Expression>r1 = s1.substring(2,4);</Expression>
            <Range>
                <Min />

```

```

        <Max />
    </Range>
</Computation>
</Reading>
<Reading id="r2">
    <Type>Physical</Type>
    <Measurement>Diastolic</Measurement>
    <Unit>mmHg</Unit>
    <Computation>
        <Type>Formula</Type>
        <Expression>r2 = s1.substring(4,6);</Expression>
        <Range>
            <Min />
            <Max />
        </Range>
    </Computation>
</Reading>
<Reading id="r3">
    <Type>Physical</Type>
    <Measurement>Pulse</Measurement>
    <Unit>perMin</Unit>
    <Computation>
        <Type>Formula</Type>
        <Expression>r3 = s1.substring(6,8);</Expression>
        <Range>
            <Min />
            <Max />
        </Range>
    </Computation>
</Reading>
</Interface>
</Device>
</DDL>

```

## Appendix D: References

1. Open Healthcare Framework (OHF) Project SODA & Device Kit  
(<http://onestop.noaa3.awips.noaa.gov/ndfd/index.html>)
2. Sensor Model Language (SensorML) (<http://www.opengeospatial.org/standards/sensorml>)
3. SubSim Physics XML Specification Guide  
(<http://robotics.ee.uwa.edu.au/auv/subsim/doc/xml/physics.html>)
4. Using XML in a Generic Model of Mediators  
(<http://www.infloom.com/gcaconfs/WEB/philadelphia99/constant.HTM>)
5. A. Helal, H. Yang, J. King and R. Bose, "Atlas - Architecture for Sensor Network Based Intelligent Environments," Submitted to the ACM Transactions on Sensor Networks. April 2007.
6. H. Yang and A. Helal, "Safety Enhancing Mechanisms for Pervasive Computing Systems in Intelligent Environment", In Proceedings of the Middleware Support for Pervasive Computing Workshop, held in conjunction with IEEE PerCom 2008, Hong Kong, March 2008.