The DomoBus System Specification Language

Prof. Renato Nunes

Instituto Superior Técnico

renato.nunes @ tecnico.ulisboa.pt

V2.0c - 29/02/2016

Index

1. Introduction

This document describes an XML-based specification language for DomoBus systems. The specification language can be divided into two parts:

- One that defines generic property types and, based on them, defines also generic device types; and
- A second part that defines a specific system and the physical structure of a home or building.

All the information is stored on a single file to simplify access and management of information coherence. The first part of a given specification can be copied and reused in other systems.

The current specification is quite complete and functional. In some cases in can even include details that may not be required by some applications (in which cases they can just ignore those details).

This specification is always open to improvements and, so, it can be expanded in the future. Because of this, every application that is designed to process a DomoBus specification must be prepared to ignore tags and attributes that are unknown or that are not relevant to the application. This can be done easily, due to the usage of XML, and that was one of the reasons why the "eXtended Markup Language" was chosen for the DomoBus system specification.

This document assumes the reader is familiar with the generic model of a home automation device used in DomoBus. However, for completeness reasons, a brief introduction is given in next section.

2. The DomoBus Approach

One of the key aspects of the DomoBus approach is its model of a home automation device. That model is very simple and modular, making the DomoBus a flexible and expandable system that, at the same time, offers a very good level of functionality.

2.1 Model of a home automation device

In DomoBus a home automation device is a generic entity characterized by one or more properties. The definition of any device can be done dynamically, adding as many properties as required to express the functional level needed for the device.

For example, if we want to represent a temperature sensor, we can refer to a generic entity (a device) that has only one property – the current temperature. If we want to represent an adjustable light, we can describe it using two properties: a "state" property that identifies if the light is turned on or off, and an "intensity" property that, for example, represents the light intensity in a scale from 0 to 100%. This model can easily be used to represent more complex devices such a TV. Assuming it is only relevant to control the state, channel and volume of the TV, only three properties are needed: "state" (on/off), "channel" (1-250) and "volume" (0-20). (Note: the physical interface with the TV could be implemented with a remote command emulator that emits infrared signals accordingly with the values of the device's properties; so, it can be applied to any TV; a similar approach can be used with air-conditioners and other equipments).

2.2 Interaction model

Given that home automation devices are represented in such a generic and uniform way, this allows the definition of an also simple and generic way to interact with them, which is based on only three types of messages:

- GET – Allows reading the value of a property of a given device; using this message one can, at any moment, monitor the state of any device.
- SET – Allows writing a new value to a property of a device; this offers a mechanism to change the state of a device and, so, command it.
- NOTIFY – This message is transmitted automatically by the devices whenever the value of one of its properties changes. This allows supervision-level applications or user-interface applications to be always up to date, without the need to continuously send GET messages to know the system current state. The existence of the NOTIFY messages make the communication infrastructure much more effective (reducing the number of messages) and minimizes system update latency (period of time between a change in a property and the time the system gets updated).

3. XML Conventions in this Document

3.1 The use of attributes

This XML specification follows a style where the usage of attributes is preferred over the usage of only elements, as this leads to a much more compact format. This becomes clear looking at the following examples.

Using only elements:

```
<Element1>
    <Element1ID> # </ Element1ID>
    <Element1Name> x </ Element1Name>
    <Element1Value> # </Element1Value>
</Element1>
```

Using attributes (this is the preferred way)**:**

```
<Element1 ID="#" Name="x" Value="#" />
```

3.2 Representation of attribute values

In the examples given along this document the character '**#**' will be used to represent a number and '**x**' will be used to represent an alphabetic or alphanumeric string. Please note that some exceptions may exist, especially regarding the indication of values (typically, these are numbers but, sometimes, they may be character strings or strings that represent a sequence of hexadecimal values such as "\x03\xAB\xF2\x41").

Given the last XML line shown above, a possible example of an actual line could be:

```
<Element1 ID="125" Name="Dummy Name" Value="32" />
```

3.3 Representation of lists of elements

In many situations the specification expects a list of elements. XML supports this automatically and we just need to use as many elements as required. However, to make more explicit that a list is expected, a dedicated element will always be used to indicate the beginning and ending of that list. See next example.

Lists of elements:

```
<Element1List>
    <Element1 ID=”125” Name=”Dummy Name” Value=”32” />
    <Element1 ID=”25” Name=”Name25” Value=”12” />
    <Element1 ID=”32” Name=”Name32” Value=”75” />
</Element1List>
```

Typically a list has, at least, one element. However, situations may exist where it is acceptable to have an empty list (list with no elements).

3.4 About the use of IDs

In the specification, every element will require a numeric ID to distinguish and easily identify it. This ID can be thought as a database/table key, simplifying identification and allowing other attributes to be changed at any moment (such as names), without any consequence, as long the element’s ID is kept the same.

Due to this approach, whenever it is required to reference an element, its ID will be used. An attribute that contains a reference to an element has a name that always starts with “Ref”. See next example.

ID referencing:

```
. . .
<Element1 ID=”125” Name=”Dummy Name” Value=”32” />
. . .

<Element2 ID=”27” Name=”MyName” RefElement1=”125” />
```

4. Definition of Generic Device Types

The specification of a DomoBus system includes the definition of generic “Device Types”, which can be reused in different systems (just by copying their definition).

The definition of a device type consists, essentially, in listing a set of properties that characterize the device. Each property is based on one of three predefined value types – scalar, enumerated and array. These three types of values are the only thing that is fixed and must be built in the applications.

Current specification allows also defining conversions that can be applied to values.

And device types may be organized into classes, as a means to structure the information.

All these subjects will be addressed next.

4.1 Device classes

The concept of a device class is optional and is introduced only as a means to organize devices types, when there is a complex system with a big number of different device types. A device type can belong only to a single device class.

Definition of device classes

```
<DeviceClassList>
    <DeviceClass ID="#" Name="x" />
</DeviceClassList>
```

Example:

```
<DeviceClassList>
    <DeviceClass ID="1" Name="Lighting" />
    <DeviceClass ID="2" Name="Heating" />
    <DeviceClass ID="3" Name="Security" />
</DeviceClassList>
```

4.2 Conversion of values

Each device has one or more properties and each property has a value. Values are represented, internally, by an integer (with 8 bits or 16 bits of useful information) or by an array of bytes. To allow the translation of integers and byte arrays to other types of values such as signed fixed point numbers, floating point numbers with double precision or any other type of data, it is possible to use conversion formulas and conversion objects.

Conversion formulas are represented by character sequences in which the quantity to convert is represented by the letter ‘x’. Examples: "2+x/10" and "123+2*x^3-x". Two formulas are provided for each conversion. One, identified by “UserToSystem”, is used to convert the user representation to the internal representation; the other, identified by “SystemToUser”, allows the opposite conversion.

Calculations are performed in floating point and the result is rounded and shown to the user with the number of decimal places defined by the attribute “DecimalPlaces”. If the attribute is equal to zero (no decimal places), the value is rounded to an integer.

Conversion objects can be used in more complex cases that cannot be easily expressed as a formula. Conversion objects are implemented in the application code and are referenced in the XML. This approach offers a good level of functionality but requires that the conversion algorithms be previously programmed before they can be used.

Conversion of values

```
<ConversionFormulaList>
    <ConversionFormula ID="#" Name="x"
                       UserToSystem="x" SystemToUser="x" DecimalPlaces="#" />
</ConversionFormulaList>

<ConversionObjectList>
    <ConversionObject ID="#" Name="x"
                       UserToSystemObj="#" SystemToUserObj="#" DecimalPlaces="#" />
</ConversionObjectList>
```

Examples:

```
<ConversionFormulaList>
    <ConversionFormula ID="1" Name="Doubles the value"
                        UserToSystem="2*x" SystemToUser="x/2" DecimalPlaces="0" />
    <ConversionFormula ID="2" Name="Times 10 plus 200"
                UserToSystem="10*x+200" SystemToUser="(x-200)/10" DecimalPlaces="1" />
                <!-- real value varies between -20.0 and 60.0. Value 23.4 is represented by 434 -->
</ConversionFormulaList>

<ConversionObjectList>
    <ConversionObject ID="1" Name="Intensity - lux"
                        UserToSystemObj="1" SystemToUserObj="2" DecimalPlaces="2" />
</ConversionObjectList>
```

4.3 Value types

The value of a property can be of three different types:

- ScalarValueType – A “scalar” is an integer value that can be represent by 8 bits (for values from 0 to 255) or by 16 bits (for values from 0 to 65535).
- EnumValueType – An “enumerated” correspond to pairs "name, value" (value is represented by an 8 bit quantity).
- ArrayValueType – Array of bytes.

Scalars can have a minimum and a maximum value, and may define a step value for increments and decrements. They may also identify units for the quantity represented.

Scalar values and arrays allow indication of a conversion mechanism, which can be of type “FORMULA” or “OBJECT”. Arrays support only object conversions, as they can represent very different types of data, such as common strings or floating point numbers.

Value types

```
<ScalarValueTypeList>
    <ScalarValueType ID="#" Name="x" NumBits="#" Units=”x” MinValue="#" MaxValue="#" Step="#">
        <ValueConversion Type="x" Ref="#" />
            <!-- Type can be “FORMULA” or “OBJECT” -->
            <!-- The element “ValueConversion” is optional; use only when a conversion is required -->
    </ScalarValueType>
</ScalarValueTypeList>

<EnumValueTypeList>
    <EnumValueType ID="#" Name="x">
        <Enumerated Name="x" Value="#" />
        <Enumerated Name="x" Value="#" />
    </EnumValueType>
</EnumValueTypeList>

<ArrayValueTypeList>
    <ArrayValueType ID="#" Name="x" MaxLen="x">
        <ValueConversion Type="OBJECT" Ref="#" />
            <!-- Arrays may only be associated with the “OBJECT” conversion type -->
            <!-- The element “ValueConversion” is optional; use only when a conversion is required -->
    </ArrayValueType>
</ArrayValueTypeList>
```

Examples:

```
<ScalarValueTypeList>
    <ScalarValueType ID="1" Name="Percentage (0-100)"  NumBits="8"
                                        Units="%" MinValue="0" MaxValue="100" Step="1">
        <!-- no value conversion -->
    </ScalarValueType>
    <ScalarValueType ID="2" Name="Power" NumBits="16"
                                        Units="Watt" MinValue="0" MaxValue="1000" Step="10">
        <ValueConversion Type="FORMULA" Ref="2" />
    </ScalarValueType>
</ScalarValueTypeList>

<EnumValueTypeList>
    <EnumValueType ID="1" Name="On-Off">
        <Enumerated Name="Off" Value="0" />
        <Enumerated Name="On" Value="1" />
    </EnumValueType>
    <EnumValueType ID="2" Name="Air Conditioning Commands">
        <Enumerated Name="Off" Value="0" />
        <Enumerated Name="Heating" Value="1" />
        <Enumerated Name="Cooling" Value="2" />
    </EnumValueType>
</EnumValueTypeList>

<ArrayValueTypeList>
    <ArrayValueType ID="1" Name="Company name" MaxLen="10">
        <!-- no value conversion -->
    </ArrayValueType>
    <ArrayValueType ID="2" Name="Float IEEE" MaxLen="8">
        <ValueConversion Type="OBJECT" Ref="1" />
    </ArrayValueType>
</ArrayValueTypeList>
```

4.4 Device types

Devices are characterized by a collection of properties. Each property has a name and an access mode that can be: “RW” – Read and Write, “RO” – Read Only, and “WO” – Write Only. Each property identifies also its value type, which can be "SCALAR", "ENUM" or "ARRAY", and references the correspondent value type, which must have been defined previously (see section 4.3).

Device types

```
<DeviceTypeList>
    <DeviceType ID="#" Name="x" RefDeviceClass="#" Description="x">
        <PropertyList>
            <Property ID="#" Name="x" AccessMode="x" ValueType="x" RefValueType="#" />
        </PropertyList>
    </DeviceType>
</DeviceTypeList>
```

Example:

```
<DeviceTypeList>
    <DeviceType ID="1" Name="Adjustable light" RefDeviceClass="1" Description="-">
        <PropertyList>
            <Property ID="1" Name="On-Off" AccessMode="RW"
                                                ValueType="ENUM" RefValueType="1" />
            <Property ID="2" Name="Intensity" AccessMode="RW"
                                                ValueType="SCALAR" RefValueType="1" />
        </PropertyList>
    </DeviceType>

    <DeviceType ID="2" Name="Temperature Sensor" RefDeviceClass="2" Description="-">
        <PropertyList>
            <Property ID="3" Name="Temperature" AccessMode="RO"
                                                ValueType="SCALAR" RefValueType="1" />
        </PropertyList>
    </DeviceType>
</DeviceTypeList>
```

5. Specification of an Actual System

The specification of an actual system involves the definition of a home's structure, the definition of available services and the definition of existent devices. The specification includes also information about system users and their access privileges.

5.1 Definition of users and access levels

A system can be accessed by many users. Each user will have an identifier, a name, a password and an access level. Users can only access devices that have an access level smaller or equal to the user level.

If a user can access a device, he can block it. A blocked device can only be unblocked by the user who blocked it or by someone else with a greater access level than that user.

The access control mechanism just described is simple but allows a good level of functionality. In the future, new and more complex mechanisms may be developed.

In the current approach, the number of access levels is fixed and vary from 0 up to 9. Level 0 is the one with fewer privileges. The upper value (9) can be increased, although it is not foreseen the need for doing so. All values within the range 0 – 9 are valid and absolute (level 5 is always greater than level 4).

To simplify the usage of the access levels, there is a section in the specification that allows the association of a name for each level. If a level has no name defined, it will use a default name such as "level_8" for the 8th level.

Users and access levels

```
<AccessLevelList>
    <AccessLevel Level="#" Name="x" />
</AccessLevelList>


<UserList>
    <User ID="#" Name="x" Password="x" AccessLevel="#" />
</UserList>
```

Note: The attributes "Name" and "Password" should be encrypted for security reasons.

Examples:

```
<AccessLevelList>
    <AccessLevel Level="0" Name="Free Access" />
    <AccessLevel Level="1" Name="Guest" />
    <AccessLevel Level="2" Name="Common User - Child" />
    <AccessLevel Level="4" Name="Common User - Parent" />
    <AccessLevel Level="9" Name="Administrator" />
</AccessLevelList>


<UserList>
    <User ID="1" Name="abcd" Password="efgh" AccessLevel="2" />
    <User ID="2" Name="ijkl" Password="mnop" AccessLevel="9" />
</UserList>
```

5.2 Definition of the structure of a house

A house may have one or more floors (at least one floor must exist). Each floor may have different divisions (rooms or areas).

A floor has a name and an attribute that is used for ordering purposes. That attribute allows specifying the relative order of the floors in height. For example, an attic will have a value greater than a ground floor, and this will have a greater value than a basement.

Each division (room or area) has a name, a reference to the floor where it is located and an access level. This access level will be used as a default value for the access level of the devices that are located in the division.

In the described structure, each division has a unique identifier, independently of the floor it belongs to. This was selected to allow a direct access to divisions, instead of requiring accessing first to a floor and then to a list of associated divisions.

Despite the names used in the tags, this section of the XML specification can also be used to identify other spaces such as a garden, a pool, a lawn or an outside garage.

This specification was adopted due to its simplicity. Note that it contains only information about the composition of the space. It does not support the definition of vicinity or connectivity between the different divisions and different floors. These relations may be needed for some applications, in which case the current specification must be extended.

House structure

```
<House ID="#" Name="x" Address="x" Phone="x">
    <FloorList>
        <Floor ID="#" Name="x" HeightOrder="#" />
    </FloorList>
    <DivisionList>
        <Division ID="#" Name="x" RefFloor="#" AccessLevel="#" />
    </DivisionList>
</House>
```

Example:

```
<House ID="1" Name="Villa Silva" Address="xxxxx" Phone="xxxx">
    <FloorList>
        <Floor ID="1" Name="Basement" HeightOrder="-1" />
        <Floor ID="2" Name="Ground-floor" HeightOrder="0" />
        <Floor ID="3" Name="Attic" Order="1" />
    </FloorList>
    <DivisionList>
        <Division ID="1" Name="Basement" RefFloor=”1” AccessLevel="1" />
        <Division ID="2" Name="Kitchen" RefFloor=”2”  AccessLevel="3" />
        <Division ID="3" Name="Living-room" RefFloor=”2” AccessLevel="2" />
        <Division ID="4" Name="Hall" RefFloor=”2” AccessLevel="2" />
        <Division ID="5" Name="Attic" RefFloor=”3” AccessLevel="2" />
    </DivisionList>
</House>
```

5.3 Definition of services

The concept of service is used to group devices that are related functionally. To make this mechanism more flexible it is possible for a device to belong to more than one service. As an example, consider a movement detector, which can be associated to both the security service (it can be used to detect an intrusion) and to the lighting service (to automate lighting when someone is detected).

Definition of services

```
<ServiceList>
    <Service ID="#" Name="x" />
</ServiceList>
```

Example:

```
<ServiceList>
    <Service ID="1" Name="Heating" />
    <Service ID="2" Name="Lighting" />
    <Service ID="3" Name="Security” />
    <Service ID="4" Name="Appliances" />
    <Service ID="5" Name="Irrigation" />
</ServiceList>
```

5.4 Definition of home automation devices

The actual devices present in a system are defined in this section of the specification. Each device has a name and references its type, which was defined previously and details its characteristics. The specification of a device includes also its communication address, location (division), access level and a list of services it is associated with. The device’s access level discriminates both monitoring and command operations, which are indicated in the same attribute using two fields separated by a comma. The first field regards the access level for reading the device’s properties and the second field regards the access level for writing them (which translates into commanding the device).

A user can block the access to a device (for reading, writing or both) as long as he has an access level greater or equal to the access level of the device. When a device is blocked it can only be unblocked by the same user or by someone that has a greater access level than that user. The attribute "UserBlocked" contains two fields separated by a comma; the first regards reading/monitoring access and the second regards writing/actuation access. Each field may contain '-' (not blocked) or the user ID that blocked the device. Given a user ID the system can know its access level.

Device definition

```
<DeviceList>
    <Device ID="#" RefDeviceType="#" Name="x"
                            Address="#" RefDivision="#" AccessLevel="#,#" UserBlocked="#,#">
        <DeviceServiceList>
            <DeviceService RefService="#" />
        </DeviceServiceList>
    </Device>
</DeviceList>
```

Example:

```
<DeviceList>
    <Device ID="1" RefDeviceType="1" Name="Kitchen_Lamp"
                        Address="#0100" RefDivision="2" AccessLevel="3,5" UserBlocked="-,27" />
    <!-- Device unblocked for reading (-) and blocked for writing by the user with ID 27;
        only user 27 can unblock the device or someone with greater access level than that user -->
        <DeviceServiceList>
            <DeviceService RefService="1" />
            <DeviceService RefService="4" />
        </DeviceServiceList>
    </Device>
</DeviceList>
```

6. Monitoring and Commanding a System – The User Interface

Using the specification described in this document it is possible to develop user interface applications that are generic and work with any system. Applications will read the XML specification of a given system and house, and adapt automatically to its content. User interface applications should allow, at least, two navigation schemes – spatial and functional.

6.1 Spatial navigation

In the spatial navigation mode, the user can browse the floors and divisions of the house, can select a specific division and browse the devices located there. Selecting a device allows monitoring its properties and change them (actuating over the device).

6.2 Navigation by service

The second navigation mode allows browsing through available services (for example, lighting, security, air-conditioning, window blinds and entertainment). After selecting a service the user will have access to the associated devices, and can then monitor or change their properties' values.

Mixed modes are also possible and should be used whenever it offers advantages. For example, when browsing a division that contain a large number of devices, it could be helpful to restrict the devices shown by selecting a specific service. In this way navigation can be faster and more user friendly.

6.3 Favorites

The user should have the possibility to indicate which divisions and devices are used more frequently, in order to allow a more direct access to them. So, the application can have a "Favorites" area where the user can add (and remove) divisions or individual devices, for a faster access. The favorite's data is unique for each user.

Definition of favorites

```
<Favorites>
    <FavoriteList RefUser="#">
        <FavoriteDivision ID="#" RefDivision="#" />
        <FavoriteDevice ID="#" RefDevice="#" />
    </FavoriteList>
</Favorites>
```

Example:

```
<Favorites>
    <FavoriteList RefUser="1">
        <FavoriteDevice ID="1" RefDevice="7" />
        <FavoriteDevice ID="2" RefDevice="5" />
    </FavoriteList>
    <FavoriteList RefUser="2">
        <FavoriteDivision ID="1" RefDivision="2" />
        <FavoriteDevice ID="2" RefDevice="5" />
    </FavoriteList>
</Favorites>
```

6.4 Alarms

Safety and security are very important in a house. It should be possible to specify relevant devices that will generate events or alarms when they become active (which may correspond, for example, to a gas leak, fire, water leak, window open, door open, oven switched on for too long, etc). Those events or alarms should be displayed in the user interface, in a well defined place, allowing the users to easily see them and quickly access relevant information about them.

6.5 Scenarios

Sometimes the user may want to set multiple devices at once. With the mechanisms described so far, the user would have to access every device, access the relevant properties and change them, one by one, which is not very practical. To circumvent this, the notion of scenario is introduced. A scenario has a name, such as "See TV", "Go to bed", "Going out" or "Weekend", which should reflect the situation in which it is applied. And each scenario includes a list of actions that set the desired state to the devices involved in the scenario. Each action defines a value to write to a device's property. When a scenario is activated, all the associated actions will be executed.

Definition of scenarios

```
<ScenarioList>
    <Scenario ID="#" Name="x">
        <ActionList>
            <Action ID="#" Name="x" RefDevice="#" RefProperty="#" Value="x" />
        </ActionList>
    </Scenario>
</ScenarioList>
```

Example:

```
<ScenarioList>
    <Scenario ID="1" Name="See TV">
        <ActionList>
            <Action ID="1" Name="Close blinds" RefDevice="1" RefProperty="0" Value="0" />
            <Action ID="2" Name="Lamp at 50%" RefDevice="5" RefProperty="1" Value="50" />
            <Action ID="3" Name="Ceiling lamp at 40%" RefDevice="9" RefProperty="1" Value="40" />
        </ActionList>
    </Scenario>
</ScenarioList>
```

7. Specifying the State of a System

Most applications need to store the current state of the devices they manage or interact with. The state of a device is constituted by the values of its properties.

The DomoBus system enforces an approach where every device must inform whenever there is a change in one of its properties (which is done using NOTIFY messages). And there is a "publish-subscribe" mechanism that allows the new values to be propagated to all applications that are interested in them. Keeping in memory the state of the devices allow a fast access to it, avoiding unnecessary queries to the devices themselves. Note that, because of the notify mechanism referred above, the applications will have always an updated view of the devices' state they subscribed to.

In some situations, mainly when simulating devices or for testing purposes, an application may require storing the current state of various devices, to be able to read it back, later on. In those cases, it is suggested to use XML to store the devices' state as illustrated below.

The attribute "InvalidValue" is optional and may be equal to TRUE or FALSE. If the attribute is not present it is equivalent to have InvalidValue="FALSE". When InvalidValue="TRUE" it states that the current property's value is unknown or there is some problem with the device and the property's value should be ignored.

Specifying the state of a system

```
<DeviceStateList>
    <DeviceState RefDevice="#" RefProperty="#" Value="x" InvalidValue="TRUE or FALSE" />
</DeviceStateList>
```

Example:

```
<DeviceStateList>
    <DeviceState RefDevice="1" RefProperty="0" Value="10" />
    <DeviceState RefDevice="1" RefProperty="1" Value="2" />
    <DeviceState RefDevice="2" RefProperty="0" Value="0" InvalidValue="FALSE " />
    <DeviceState RefDevice="3" RefProperty="2" Value="150" InvalidValue="TRUE" />
</DeviceStateList>
```

8. Conclusion

This document presented the specification language that is used to describe DomoBus systems. The specification language is very flexible and allows the description of any system, with any combination of devices, and also allows the description of the house structure where the system is installed.

The current specification language allows the definition of property types and, based on them, the definition of device types. This information can be reused in different systems and can be extended whenever required. Based on available device types, the actual devices of a system can be defined.

The specification includes also the definition of users and introduces a simple access control mechanism based on access levels. Every device can have two different access levels: one for reading (to monitor a device's state) and other for writing (to change a device's state).

The use of this specification language allows de implementation of generic applications that can work with any system. This specification can evolve and new elements and attributes can be added at any moment. Applications must be able to deal with this, reading and processing only the elements and attributes relevant to them, and ignoring all other information that may be present in a given system specification.