XEP-0325: Internet of Things - Control

Peter Waher
mailto:peterwaher@hotmail.com
xmpp:peter.waher@jabber.org
http://www.linkedin.com/in/peterwaher

2017-05-20
Version 0.5

Status	Type	Short Name
Retracted	Standards Track	sensor-network-control

Note: This specification has been retracted by the author; new implementations are not recommended.
This specification describes how to control devices or actuators in an XMPP-based sensor network.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

Actuators are devices in sensor networks that can be controlled through the network and act with the outside world. In sensor networks and Internet of Things applications, actuators make it possible to automate real-world processes. This document defines a mechanism whereby actuators can be controlled in XMPP-based sensor networks, making it possible to integrate sensors and actuators of different brands, makes and models into larger Internet of Things applications.
Note has to be taken, that these XEP's are designed for implementation in sensors, many of which have very limited amount of memory (both RAM and ROM) or resources (processing power). Therefore, simplicity is of utmost importance. Furthermore, sensor networks can become huge, easily with millions of devices in peer-to-peer networks.
Sensor networks contains many different architectures and use cases. For this reason, the sensor network standards have been divided into multiple XEPs according to the following table:

XEP	Description
xep-0000-IoT-BatteryPoweredSensors	Defines how to handle the peculiars related to battery powered devices, and other devices intermittently available on the network.
xep-0000-IoT-Discovery	Defines the peculiars of sensor discovery in sensor networks. Apart from discovering sensors by JID, it also defines how to discover sensors based on location, etc.
xep-0000-IoT-Events	Defines how sensors send events, how event subscription, hysteresis levels, etc., are configured.
xep-0000-IoT-Interoperability	Defines guidelines for how to achieve interoperability in sensor networks, publishing interoperability interfaces for different types of devices.
xep-0000-IoT-Multicast	Defines how sensor data can be multicast in efficient ways.
xep-0000-IoT-PubSub	Defines how efficient publication of sensor data can be made in sensor networks.
xep-0000-IoT-Chat	Defines how human-to-machine interfaces should be constructed using chat messages to be user friendly, automatable and consistent with other IoT extensions and possible underlying architecture.
XEP-0322	Defines how to EXI can be used in XMPP to achieve efficient compression of data. Albeit not a sensor network specific XEP, this XEP should be considered in all sensor network implementations where memory and packet size is an issue.

XEP	Description
XEP-0323	Provides the underlying architecture, basic operations and data structures for sensor data communication over XMPP networks. It includes a hardware abstraction model, removing any technical detail implemented in underlying technologies. This XEP is used by all other sensor network XEPs.
XEP-0324	Defines how provisioning, the management of access privileges, etc., can be efficiently and easily implemented.
XEP-0325	This specification. Defines how to control actuators and other devices in Internet of Things.
XEP-0326	Defines how to handle architectures containing concentrators or servers handling multiple sensors.
XEP-0331	Defines extensions for how color parameters can be handled, based on Data Forms (XEP-0004) XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>.
XEP-0336	Defines extensions for how dynamic forms can be created, based on Data Forms (XEP-0004) XEP-0004: Data Forms <https://xmpp.org/extensions/xep-0004.html>., Data Forms Validation (XEP-0122) XEP-0122: Data Forms Validation <https://xmpp.org/extensions/xep-0122.html>., Publishing Stream Initiation Requests (XEP-0137) XEP-0137: Publishing Stream Initiation Requests <https://xmpp.org/extensions/xep-0137.html>. and Data Forms Layout (XEP-0141) XEP-0141: Data Forms Layout <https://xmpp.org/extensions/xep-0141.html>..

2 Glossary

The following table lists common terms and corresponding descriptions.

Actuator Device containing at least one configurable property or output that can and should be controlled by some other entity or device.

Computed Value A value that is computed instead of measured.

Concentrator Device managing a set of devices which it publishes on the XMPP network.

Field One item of sensor data. Contains information about: Node, Field Name, Value, Precision, Unit, Value Type, Status, Timestamp, Localization information, etc. Fields should be unique within the triple (Node ID, Field Name, Timestamp).

Field Name Name of a field of sensor data. Examples: Energy, Volume, Flow, Power, etc.

Field Type What type of value the field represents. Examples: Momentary Value, Status Value, Identification Value, Calculated Value, Peak Value, Historical Value, etc.

Historical Value A value stored in memory from a previous timestamp.

Identification Value A value that can be used for identification. (Serial numbers, meter IDs, locations, names, etc.)

Localization information Optional information for a field, allowing the sensor to control how the information should be presented to human viewers.

Meter A device possible containing multiple sensors, used in metering applications. Examples: Electricity meter, Water Meter, Heat Meter, Cooling Meter, etc.

Momentary Value A momentary value represents a value measured at the time of the readout.

Node Graphs contain nodes and edges between nodes. In Internet of Things, sensors, actuators, meters, devices, gatewats, etc., are often depicted as nodes whereas links between sensors (friendships) are depicted as edges. In abstract terms, it's easier to talk about a Node, rather than list different possible node types (sensors, actuators, meters, devices, gateways, etc.). Each Node has a Node ID.

Node ID An ID uniquely identifying a node within its corresponding context. If a globally unique ID is desired, an architecture should be used using a universally accepted ID scheme.

Parameter Readable and/or writable property on a node/device. The XEP-0326 Internet of Things - Concentrators (XEP-0326) XEP-0326: Internet of Things - Concentrators <https://xmpp.org/extensions/xep-0326.html>. deals with reading and writing parameters on nodes/devices. Fields are not parameters, and parameters are not fields.

Peak Value A maximum or minimum value during a given period.

Precision In physics, precision determines the number of digits of precision. In sensor networks however, this definition is not easily applicable. Instead, precision determines, for example, the number of decimals of precision, or power of precision. Example: 123.200 MWh contains 3 decimals of precision. All entities parsing and delivering field information in sensor networks should always retain the number of decimals in a message.

Sensor Device measuring at least one digital value (0 or 1) or analog value (value with precision and physical unit). Examples: Temperature sensor, pressure sensor, etc. Sensor values are reported as fields during read-out. Each sensor has a unique Node ID.

SN Sensor Network. A network consisting, but not limited to sensors, where transport and use of sensor data is of primary concern. A sensor network may contain actuators, network applications, monitors, services, etc.

Status Value A value displaying status information about something.

Timestamp Timestamp of value, when the value was sampled or recorded.

Token A client, device or user can get a token from a provisioning server. These tokens can be included in requests to other entities in the network, so these entities can validate access rights with the provisioning server.

Unit Physical unit of value. Example: MWh, l/s, etc.

Value A field value.

Value Status Status of field value. Contains important status information for Quality of Service purposes. Examples: Ok, Error, Warning, Time Shifted, Missing, Signed, etc.

Value Type Can be numeric, string, boolean, Date & Time, Time Span or Enumeration.

WSN Wireless Sensor Network, a sensor network including wireless devices.

XMPP Client Application connected to an XMPP network, having a JID. Note that sensors, as well as applications requesting sensor data can be XMPP clients.

3 Use Cases

Control in sensor networks is about setting output values. To make the implementation simple, it is assumed that control of a device can be made using a single message. If only a simple set operation is requested, a <message> stanza can be sent. If an acknowledgement (ACK) of the operation (or Not-acknowledgement NACK) of the operation is desired, an <iq> stanza can be used instead.
To set control parameters in a device, the **set** command is sent to the device. The set command allows for two different ways of setting control parameters:

- Using strongly typed parameters. This way performs best in EXI compression and automation.
- Using a weakly typed data form. This might be better for manually setting control parameters, since it allows the device to give the user a better user interface explaining available control parameters.

What type of control parameters there are available in different types of devices is described in [1].
If the device is a concentrator, as defined in Internet of Things - Concentrators, an handles multiple nodes behind it, which node(s) to control is defined using **node** elements. If not a concentrator, the use of **node** elements is not necessary, and control commands are sent directly to the device itself.

3.1 Control commands

3.1.1 Sending a control command using a message stanza

Following is an example of a control command sent using a message stanza:

Listing 1: Message stanza for setting a value

```
<message from='master@example.org/amr'
  to='digital.output@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <boolean name='Output' value='true'/>
  </set>
</message>
```

Note that any response is supressed when sending a message stanza, regardless if the desired control command could be executed or not. The following example shows how the same control command could be issued using an IQ stanza instead:

3.1.2 Sending a control command using an IQ stanza

Following is an example of a control command sent using an iq stanza:

Listing 2: IQ stanza for setting a value

```
<iq type='set'
  from='master@example.org/amr'
  to='digital.output@example.org'
  id='1'>
  <set xmlns='urn:xmpp:iot:control' xml:lang='en'>
    <boolean name='Output' value='true'/>
  </set>
</iq>

<iq type='result'
  from='digital.output@example.org'
  to='master@example.org/amr'
```

[1] XEP-xxxx: Internet of Things - Interoperability <xep-0000-IoT-Interoperability.html>

```
  id='1'>
  <setResponse xmlns='urn:xmpp:iot:control'/>
</iq>
```

Note: An empty **setResponse** element means that the control command was executed as provided in the request. Sometimes, the device can restrict the command to a subset of nodes and/or parameters. In such cases, the **setResponse** element will contain what nodes and/or parameters were finally used to perform the command. For examples of this, see Internet of Things - Provisioning (XEP-0324) [2].
In the following use cases, often a message stanza will be used to illustrate the point. However, the same operation could equally well be used using an iq stanza instead.

3.1.3 Control failure response

By using an IQ stanza, the caller can receive an acknowledgement of the reception of the command, or error information if the command could not be processed. Following is an example of a control command sent using an iq stanza, where the receiver reports an error back to the caller:

Listing 3: Control failure response

```
<iq type='set'
  from='master@example.org/amr'
  to='analog.output@example.org'
  id='2'>
  <set xmlns='urn:xmpp:iot:control' xml:lang='en'>
    <boolean name='Output' value='true'/>
  </set>
</iq>

<iq type='error'
  from='analog.output@example.org'
  to='master@example.org/amr'
  id='2'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <paramError xmlns='urn:xmpp:iot:control' var='Output'>Invalid
       parameter type.</error>
  </error>
</iq>
```

Here, the **paramError** element is used in the IQ Error response, to provide error information related to a specific control parameter.

[2] XEP-0324: Internet of Things - Provisioning <https://xmpp.org/extensions/xep-0324.html>.

3.2 Setting control parameters

The following sub-sections illustrate how to set parameters of different types in a device.

3.2.1 Setting a single boolean-valued control parameter

Setting single boolean-valued control parameters is a common use case, for instance when controlling digital outputs. The following example shows how a boolean value can be set in a device.

Listing 4: Setting a single boolean-valued control parameter

```
<message from='master@example.org/amr'
  to='digital.output@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <boolean name='Output' value='true'/>
  </set>
</message>
```

3.2.2 Setting a single 32-bit integer-valued control parameter

Setting single integer-valued control parameters is a common use case, for instance when controlling analog outputs. The following example shows how a 32-bit integer value can be set in a device.

Listing 5: Setting a single 32-bit integer-valued control parameter

```
<message from='master@example.org/amr'
  to='analog.output@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <int name='Output' value='50000'/>
  </set>
</message>
```

3.2.3 Setting a single 64-bit integer-valued control parameter

Setting single integer-valued control parameters is a common use case, for instance when controlling analog outputs. Even though 32-bit integers may cover most control needs, it might in some cases be limiting. Therefore, a 64-bit control parameters can be created. The following example shows how a 64-bit integer value can be set in a device.

Listing 6: Setting a single 64-bit integer-valued control parameter

```
<message from='master@example.org/amr'
  to='megaprecision.analog.output@example.org'>
```

```
  <set xmlns='urn:xmpp:iot:control'>
    <long name='Output' value='500000000000000'/>
  </set>
</message>
```

3.2.4 Setting a single string-valued control parameter

Setting single string-valued control parameters is a common use case, for instance when controlling text displays. The following example shows how a string value can be set in a device.

Listing 7: Setting a single string-valued control parameter

```
<message from='master@example.org/amr'
  to='text.display@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <string name='Row1' value='Temperature: 21.4 °C'/>
  </set>
</message>
```

3.2.5 Setting a single double-valued control parameter

Setting single double-valued control parameters can be an alternative form of controlling analog outputs for instance. The following example shows how a double value can be set in a device.

Listing 8: Setting a single double-valued control parameter

```
<message from='master@example.org/amr'
  to='analog.output2@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <double name='4-20mA' value='8.192'/>
  </set>
</message>
```

3.2.6 Setting a single date-valued control parameter

Setting date-valued control parameters might be necessary when timing is an issue. Often it forms part of a larger context. The following example shows how a date value can be set in a device.

Listing 9: Setting a single date-valued control parameter

```
<message from='master@example.org/amr'
  to='alarm@example.org'>
```

```
  <set xmlns='urn:xmpp:iot:control'>
    <date name='TariffStartDate' value='2013-05-01'/>
  </set>
</message>
```

3.2.7 Setting a single time-valued control parameter

Setting time-valued control parameters might be necessary when timing is an issue. Often it forms part of a larger context. The following example shows how a time value can be set in a device.

Listing 10: Setting a single time-valued control parameter

```
<message from='master@example.org/amr'
  to='alarm@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <time name='Alarm␣Time' value='08:00:00'/>
  </set>
</message>
```

3.2.8 Setting a single date & time-valued control parameter

Setting date & time-valued control parameters might be necessary when timing is an issue. Often it forms part of a larger context. The following example shows how a date & time value can be set in a device.

Listing 11: Setting a single date & time-valued control parameter

```
<message from='master@example.org/amr'
  to='alarm@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <dateTime name='Alarm␣Time' value='2013-04-02T08:00:00'/>
  </set>
</message>
```

3.2.9 Setting a single duration-valued control parameter

Setting duration-valued control parameters might be necessary when timing is an issue. Often it forms part of a larger context. The following example shows how a duration value can be set in a device.

Listing 12: Setting a single duration-valued control parameter

```
<message from='master@example.org/amr'
  to='alarm@example.org'>
```

```
  <set xmlns='urn:xmpp:iot:control'>
    <duration name='Alarm_Duration' value='PT3M30S'/>
  </set>
</message>
```

3.2.10 Setting a single color-valued control parameter

Setting single color values in a device can occur in instances where color or lighting is important. Sometimes color is set using enumerations (string-valued or integer-valued parameters), and sometimes as a color property. The following example shows how a color value can be set in a device.

Listing 13: Setting a single color-valued control parameter

```
<message from='master@example.org/amr'
  to='spotlight@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <color name='Color' value='3399FF'/>
  </set>
</message>
```

3.2.11 Setting multiple control parameters at once

Often, setting a single control parameter is not sufficient for a control action. In these cases, setting multiple control parameters at once is necessary. The **set** command makes this easy however, since it allows for any number of control parameters to be set at once, as the following example shows:

Listing 14: Setting multiple control parameters at once

```
<message from='master@example.org/amr'
  to='dimmer@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <int name='FadeTimeMilliseconds' value='500'/>
    <int name='OutputPercent' value='10'/>
  </set>
</message>
```

Sometimes the order of control parameters are important in the device, and sometimes the parameters form part of a whole. It depends on the context of the device. In the above example, the order is important. When the OutputPercent control parameter is set, it will start to fade in or out to the desired setting (10%), using the fade time set previously. If the FadeTimeMilliseconds control parameter would have been set after the OutputPercent parameter, the fading would have been started using the previous setting, which might be unknown.

The order of control parameters to use depends on the device. The Control Form lists available control parameters of the device in the order they are expected to be sent to the device. The XEP xep-0000-IoT-Interoperability details what control parameters must be available for different interfaces, and if the order of control parameters is important.

3.3 Control forms

3.3.1 Getting a control form

A client can get a control form containing available control parameters of the device. This is done using the **getForm** command, as is shown in the following example:

Listing 15: Getting a control form

```
<iq type='get'
  from='master@example.org/amr'
  to='dimmer@example.org'
  id='3'>
  <getForm xmlns='urn:xmpp:iot:control' xml:lang='en'/>
</iq>

<iq type='result'
  from='dimmer@example.org'
  to='master@example.org/amr'
  id='3'>
  <x type='form'
    xmlns='jabber:x:data'
    xmlns:xdv='http://jabber.org/protocol/xdata-validate'
    xmlns:xdl:='http://jabber.org/protocol/xdata-layout'
    xmlns:xdd:='urn:xmpp:xdata:dynamic'>
    <title>Dimmer</title>
    <xdl:page label='Output'>
      <xdl:fieldref var='FaceTimeMilliseconds'/>
      <xdl:fieldref var='OutputPercent'/>
      <xdl:fieldref var='MainSwitch'/>
    </xdl:page>
    <field var='xdd␣session' type='hidden'>
      <value>325ED0F3-9A9A-45A4-9634-4E0D41C5EA06</value>
    </field>
    <field var='FadeTimeMilliseconds' type='text-single' label='Fade␣
       Time␣(ms):'>
      <desc>Time in milliseconds used to fade the light to the desired
          level.</desc>
      <value>300</value>
      <xdv:validate datatype='xs:int'>
        <xdv:range min='0' max='4095'/>
      </xdv:validate>
```

```
      <xdd:notSame/>
    </field>
    <field var='OutputPercent' type='text-single' label='Output (%):'>
      <desc>Dimmer output, in percent.</desc>
      <value>100</value>
      <xdv:validate datatype='xs:int'>
        <xdv:range min='0' max='100'/>
      </xdv:validate>
      <xdd:notSame/>
    </field>
    <field var='MainSwitch' type='boolean' label='Main switch'>
      <desc>If the dimmer is turned on or off.</desc>
      <value>true</value>
      <xdd:notSame/>
    </field>
  </x>
</iq>
```

IMPORTANT: The device MUST mark all control parameters in the form as **notSame**, as defined in [3]. If an end user would open the control form and press OK (submitting the form) without having entered a value, no value would be written, and no action taken. If only a few parameter would be edited, only those parameters would be sent to the device and only the corresponding actions taken.
All parameters in the form MUST also have validation rules defined according to XEP-0122, specifically validation data types and ranges where appropriate. This to give type information to the client, which the client later can use to send typed control commands directly, without the need to get and send data forms to the device to control it.
Also, the device SHOULD group control parameters that should be written together using pages and sections, as described in XEP-0141. Parameters MUST also be ordered in a way so that when set in that order using the typed commands, the corresponding control actions can be successfully executed.
Note: There's a difference between node parameters, as described in XEP-0326 Internet of Things - Concentrators, and control parameters as described in this document. For more information about this, please see Difference between node parameters and node control parameters.

3.3.2 Getting a control form, Failure

A device can reject a control form request. It does this returning an **error** iq stanza, as is shown in the following example:

Listing 16: Getting a control form, Failure

```
<iq type='get'
```

[3] XEP-0336: Dynamic Data Forms <http://xmpp.org/extensions/xep-0336.html>

```
  from='master@example.org/amr'
  to='dimmer@example.org'
  id='4'>
  <getForm xmlns='urn:xmpp:iot:control' xml:lang='en'/>
</iq>

<iq type='error'
  from='dimmer@example.org'
  to='master@example.org/amr'
  id='4'>
  <error type='cancel'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <text xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' xml:lang='en'>
        Access denied.</text>
  </error>
</iq>
```

3.3.3 Setting a (partial) control form

Control actions can be requested by submitting a full or partial control form back to the device. Control parameters not edited MUST not be included in the form, and the device in turn MUST ONLY invoke control actions corresponding to the parameters returned in the form.
The following example shows how control actions can be requested submitting a control parameters form to the device:

Listing 17: Setting a (partial) control form

```
<iq type='set'
  from='master@example.org/amr'
  to='dimmer@example.org'
  id='5'>
  <set xmlns='urn:xmpp:iot:control' xml:lang='en'>
    <x type='submit' xmlns='jabber:x:data'>
      <field var='xdd_session' type='hidden'>
        <value>325ED0F3-9A9A-45A4-9634-4E0D41C5EA06</value>
      </field>
      <field var='FadeTimeMilliseconds' type='text-single'>
        <value>500</value>
      </field>
      <field var='OutputPercent' type='text-single'>
        <value>10</value>
      </field>
    </x>
  </set>
</iq>

<iq type='result'
```

```
  from='dimmer@example.org'
  to='master@example.org/amr'
  id='5'>
  <setResponse xmlns='urn:xmpp:iot:control' />
</iq>
```

In this example, the FadeTimeMilliseconds and OutputPercent control parameters are sent, while the MainSwitch control parameter is left as is. Fading is therefore parformed only if the dimmer is switched on.

3.3.4 Setting a (partial) control form, Failure

A device can reject a control form submission. It does this returning an **error** iq stanza. If there are errors in the form, details are listed using **paramError** elements in the response, as is shown in the following example:

Listing 18: Setting a (partial) control form, Failure

```
<iq type='set'
  from='master@example.org/amr'
  to='dimmer@example.org'
  id='6'>
  <set xmlns='urn:xmpp:iot:control' xml:lang='en'>
    <x type='submit' xmlns='jabber:x:data'>
      <field var='xdd_session' type='hidden'>
        <value>325ED0F3-9A9A-45A4-9634-4E0D41C5EA06</value>
      </field>
      <field var='FadeTimeMilliseconds' type='text-single'>
        <value>500</value>
      </field>
      <field var='OutputPercent' type='text-single'>
        <value>200</value>
      </field>
    </x>
  </set>
</iq>

<iq type='error'
  from='dimmer@example.org'
  to='master@example.org/amr'
  id='6'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <paramError xmlns='urn:xmpp:iot:control' var='OutputPercent'>
       Invalid parameter value.</error>
  </error>
</iq>
```

3.4 Controlling devices behind a concentrator

Controlling devices behind a concentrator can be done by specifying what device(s) to control using **node** elements within the command elements sent to the concentrator. The following sub-sections show examples of how this is done.

3.4.1 Sending a control command to a node behind a concentrator

To send a control message to a specific node behind a concentrator, the **node** element can be used to identify the node, as is shown in the following example:

Listing 19: Sending a control command to a node behind a concentrator

```
<message from='master@example.org/amr'
  to='concentrator@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <node nodeId='DigitalOutput1'/>
    <boolean name='Output' value='false'/>
  </set>
</message>
```

3.4.2 Sending a control command to multiple nodes

The client can send the same control command to multiple nodes behind a concentrator by simply adding more **node** elements in the request, as is shown in the following example:

Listing 20: Sending a control command to multiple nodes

```
<message from='master@example.org/amr'
  to='concentrator@example.org'>
  <set xmlns='urn:xmpp:iot:control'>
    <node nodeId='DigitalOutput1'/>
    <node nodeId='DigitalOutput2'/>
    <node nodeId='DigitalOutput3'/>
    <node nodeId='DigitalOutput4'/>
    <boolean name='Output' value='false'/>
  </set>
</message>
```

3.4.3 Sending a control command to multiple nodes, Failure

By using an IQ stanza, the caller can receive an acknowledgement of the reception of the command, or error information if the command could not be processed. When sending a control command to multiple nodes at a time the device must validate all parameters against all nodes before taking any control action. If validation fails, an error message is returned and

no control action is taken. The following example shows an example of an erroneous control message made to multiple nodes on a device:

Listing 21: Sending a control command to multiple nodes, Failure

```
<iq type='set'
  from='master@example.org/amr'
  to='concentrator@example.org'
  id='7'>
  <set xmlns='urn:xmpp:iot:control' xml:lang='en'>
    <node nodeId='DigitalOutput1'/>
    <node nodeId='DigitalOutput2'/>
    <node nodeId='DigitalOutput3'/>
    <node nodeId='DigitalOutput4'/>
    <node nodeId='AnalogOutput1'/>
    <node nodeId='AnalogOutput2'/>
    <node nodeId='AnalogOutput3'/>
    <node nodeId='AnalogOutput4'/>
    <boolean name='Output' value='true'/>
  </set>
</iq>

<iq type='error'
  from='concentrator@example.org'
  to='master@example.org/amr'
  id='7'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <paramError xmlns='urn:xmpp:iot:control' var='Output'>Invalid
       parameter type.</error>
  </error>
</iq>
```

3.4.4 Getting a control form from multiple nodes

A client can get a control form containing available control parameters common between a set of nodes controlled by the concentrator. This is done adding a sequence of **node** elements to a **getForm** command sent to the concentrator, as is shown in the following example:

Listing 22: Getting a control form from multiple nodes

```
<iq type='get'
  from='master@example.org/amr'
  to='concentrator@example.org'
  id='8'>
  <getForm xmlns='urn:xmpp:iot:control' xml:lang='en'>
    <node nodeId='DigitalOutput1'/>
    <node nodeId='DigitalOutput2'/>
```

```
    <node nodeId='DigitalOutput3'/>
    <node nodeId='DigitalOutput4'/>
  </getForm>
</iq>

<iq type='result'
  from='concentrator@example.org'
  to='master@example.org/amr'
  id='8'>
  <x type='form'
    xmlns='jabber:x:data'
    xmlns:xdv='http://jabber.org/protocol/xdata-validate'
    xmlns:xdl:='http://jabber.org/protocol/xdata-layout'
    xmlns:xdd:='urn:xmpp:xdata:dynamic'>
    <title>DigitalOutput1, DigitalOutput2, ...</title>
    <xdl:page label='Output'>
      <xdl:fieldref var='Output'/>
    </xdl:page>
    <field var='xdd_session' type='hidden'>
      <value>325ED0F3-9A9A-45A4-9634-4E0D41C5EA06</value>
    </field>
    <field var='Output' type='boolean' label='Output'>
      <desc>If the digital output is high (checked) or low (uncheckd).
         </desc>
      <value>true</value>
      <xdd:notSame/>
    </field>
  </x>
</iq>
```

Note that only parameters that are common between the nodes defined in the request must be returned. However, all parameters must have the **notSame** flag set, regardless of current output status.

3.4.5 Getting a control form from multiple nodes, Failure

A device can reject a control form request. It does this returning an **error** iq stanza. The following example shows the device rejecting a control form request, because it does not support the handling of common parameters between multiple nodes:

Listing 23: Getting a control form from multiple nodes, Failure

```
<iq type='get'
  from='master@example.org/amr'
  to='concentrator@example.org'
  id='9'>
  <getForm xmlns='urn:xmpp:iot:control' xml:lang='en'>
```

```
    <node nodeId='DigitalOutput1'/>
    <node nodeId='DigitalOutput2'/>
    <node nodeId='DigitalOutput3'/>
    <node nodeId='DigitalOutput4'/>
  </getForm>
</iq>

<iq type='error'
  from='concentrator@example.org'
  to='master@example.org/amr'
  id='9'>
  <error type='cancel'>
    <feature-not-implemented xmlns='urn:ietf:params:xml:ns:xmpp-
        stanzas'/>
    <text xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' xml:lang='en'>
        Cannot merge control forms from different nodes.</text>
  </error>
</iq>
```

3.4.6 Setting a (partial) control form to multiple nodes

You set a control form to multiple nodes controlled by a concentrator by adding **node** elements to the **set** command sent to the concentrator, as is shown in the following example:

Listing 24: Setting a (partial) control form to multiple nodes

```
<iq type='set'
  from='master@example.org/amr'
  to='concentrator@example.org'
  id='10'>
  <set xmlns='urn:xmpp:iot:control' xml:lang='en'>
    <node nodeId='DigitalOutput1'/>
    <node nodeId='DigitalOutput2'/>
    <node nodeId='DigitalOutput3'/>
    <node nodeId='DigitalOutput4'/>
    <x type='submit' xmlns='jabber:x:data'>
      <field var='xdd_session' type='hidden'>
        <value>325ED0F3-9A9A-45A4-9634-4E0D41C5EA06</value>
      </field>
      <field var='Output' type='boolean'>
        <value>true</value>
      </field>
    </x>
  </set>
</iq>

<iq type='result'
  from='concentrator@example.org'
```

```
  to='master@example.org/amr'
  id='10'>
  <setResponse xmlns='urn:xmpp:iot:control' />
</iq>
```

3.4.7 Setting a (partial) control form to multiple nodes, Failure

A device can reject a control form submission. It does this returning an **error** iq stanza. The following example shows the device rejecting a control form submission because one of the control parameters, even though it exists on all nodes, is not of the same type on all nodes.

Listing 25: Setting a (partial) control form to multiple nodes

```
<iq type='set'
  from='master@example.org/amr'
  to='concentrator@example.org'
  id='11'>
  <set xmlns='urn:xmpp:iot:control' xml:lang='en'>
    <node nodeId='DigitalOutput1'/>
    <node nodeId='DigitalOutput2'/>
    <node nodeId='DigitalOutput3'/>
    <node nodeId='DigitalOutput4'/>
    <node nodeId='AnalogOutput1'/>
    <node nodeId='AnalogOutput2'/>
    <node nodeId='AnalogOutput3'/>
    <node nodeId='AnalogOutput4'/>
    <x type='submit' xmlns='jabber:x:data'>
      <field var='xdd␣session' type='hidden'>
        <value>325ED0F3-9A9A-45A4-9634-4E0D41C5EA06</value>
      </field>
      <field var='Output' type='boolean'>
        <value>true</value>
      </field>
    </x>
  </set>
</iq>

<iq type='error'
  from='concentrator@example.org'
  to='master@example.org/amr'
  id='11'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <paramError xmlns='urn:xmpp:iot:control' var='Output'>Invalid type
       .</error>
  </error>
</iq>
```

4 Determining Support

If an entity supports the protocol specified herein, it MUST advertise that fact by returning a feature of "urn:xmpp:iot:control" in response to Service Discovery (XEP-0030) [4] information requests.

Listing 26: Service discovery information request

```
<iq type='get'
  from='device@example.org/device'
  to='provisioning@example.org'
  id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 27: Service discovery information response

```
<iq type='result'
  from='provisioning@example.org'
  to='device@example.org/device'
  id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    ...
    <feature var='urn:xmpp:iot:control'/>
    ...
  </query>
</iq>
```

In order for an application to determine whether an entity supports this protocol, where possible it SHOULD use the dynamic, presence-based profile of service discovery defined in Entity Capabilities (XEP-0115) [5]. However, if an application has not received entity capabilities information from an entity, it SHOULD use explicit service discovery instead.

5 Implementation Notes

5.1 IQ Error Stanzas

Depending on the reason for rejecting a control request, different XMPP errors can be returned, according to the description in the following table. The table also lists recommended error type for each error. Error stanzas may also include **paramError** child elements to provide additional textual error information that corresponds to a particular control parameter provided in the request. Any custom error message not related to control parameters is returned in a **text** element.

[4] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.
[5] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.

Error Type	Error Element	Namespace	Description
cancel	forbidden	urn:ietf:params:xml:ns:xmpp-stanzas	If the caller lacks privileges to perform the action.
cancel	item-not-found	urn:ietf:params:xml:ns:xmpp-stanzas	If an item, parameter or data source could not be found.
modify	bad-request	urn:ietf:params:xml:ns:xmpp-stanzas	If the request was malformed. Examples can include trying to set a parameter to a value outside the allowed range.
cancel	feature-not-implemented	urn:ietf:params:xml:ns:xmpp-stanzas	If an action has not been implemented in the device.
wait	conflict	urn:ietf:params:xml:ns:xmpp-stanzas	If an item was locked by another user or process and could not be accessed. The operation can be retried at a later point in time.

5.2 Reading current control states

5.2.1 Using Control Forms

The simplest way to read the current control states of a device, is to request the control form of the device. The control form should contain the current values for all avilable control parameters. However, since current states might not be available at the time of the request, the states might be delivered asynchronously, using messages defined in Data Forms - Dynamic Forms (XEP-0336) [6].
Using this method has the advantage that a small actuator device does not need to implement other XEPs to support readout of current control states. If the device contains all current states readily accessible, there's no need of asynchronous updates making readout simple and straightforward.

5.2.2 Using XEP-0323 (Sensor Data)

A second option is to use Internet of Things - Sensor Data (XEP-0323) [7] to deliver current control states. Since this XEP contains mechanisms allowing for asynchronous readout of

[6] XEP-0336: Data Forms - Dynamic Forms <https://xmpp.org/extensions/xep-0336.html>.
[7] XEP-0323: Internet of Things - Sensor Data <https://xmpp.org/extensions/xep-0323.html>.

control parameter states. This makes readout of such parameters simpler. However, there's a need to map values between the two XEPs.
If a client wants to know the current status of control parameters using this method, it performs a readout of **Momentary** and **Status** values from the device, and from the returned set of values take the current control parameter value according to the following rules, ordered by priority:

- If there's a field marked as momentary value, with an unlocalized field name equal to the unlocalized control parameter name and having a compatible field value type (see table below) and a status field without the missing flag set, the value of the field should be considered the current value of the control parameter.

- If there's a field marked as status value, with an unlocalized field name equal to the unlocalized control parameter name and having a compatible field value type (see table below) and a status field without the missing flag set, the value of the field should be considered the current value of the control parameter.

- To simplify mapping, a **writable** attribute can be used to inform the client if a field name corresponds to a control parameter or not. If the attribute is available, it tells the client if it corresponds to a control parameter or not. If not available, no such deduction can be made.

Even though getting the the control form could provide the client with a quicker and easier way of retrieving control parameter values, the form is not guaranteed to contain correct current values, as described above.
The following table shows how corresponding field values should be converted to the corresponding control parameter value based on field type (x-axis) and control parameter type (y-axis). An empty cell means conversion has no meaning and types are not compatible.

325 \ 323	boolean	date	dateTime	duration	enum	int	long	numeric	string	time
boolean	x					!=0	!=0	!=0		
color									RRGGBB or RRGG-B-BAA	
date		x	Date part						(1)	
dateTime		x	x						(2)	
double	Z2					x	x	x	(3)	

325 \ 323	boolean	date	dateTime	duration	enum	int	long	numeric	string	time
duration				x					(4)	x
int	Z2				Ordinal	x	Thunk	Thunk	(5)	
long	Z2				Ordinal	x	x	Thunk	(5)	
string	xs:boolean	xs:date	xs:dateTime	xs:duration		xs:int	xs:long	(6)	x	xs:time
time			Time part	(7)					(8)	x

The following table lists notes with details on how to do conversion, if in doubt.

Note	Description
(1)	The client should try to convert the string to a date value, first according to the format specified by the XML data type xs:date, and if not possible by RFC 822.
(2)	The client should try to convert the string to a date & time value, first according to the format specified by the XML data type xs:dateTime, and if not possible by RFC 822.
(3)	The client should try to convert the string to a double-precision floating-point value, first according to the format specified by the XML data type xs:double, and if not possible using system-local string to floating-point conversion using local decimal and throusand separator settings.
(4)	The client should try to convert the string to a duration value, first according to the format specified by the XML data type xs:duration, and if not possible using the XML data type xs:time.
(5)	The client should try to convert the string to an integer value according to the corresponding XML data type formats xs:int and xs:long.
(6)	The numeric field value consists of three parts: Numeric value, number of decimals and optional unit. If no unit is provided, only the numeric value should be converted to a string (compatible with the XML data type xs:double), using exactly the number of decimals provided in the field. If a unit is provided (non-empty string) it must not be appended to the value, if the value is to be used for control output. For presentation purposes however, a space could be appended to the number and the unit appended after the space.
(7)	A duration field value contains a xs:duration value. The xs:duration has a larger domain than xs:time, and contains all xs:time values, but xs:time does not contain all possible xs:duration values. So, conversion of an xs:duration value to an xs:time value should be performed only if a duration lies between 00:00:00 and 23:59:59.
(8)	The client should try to convert the string to a time value according to the format specified by the XML data type xs:time.
x	Use the canonical conversion method.

Note	Description
Z2	true = 1, false = 0.
!=0	Nonzero = true, Zero = false.
RRGGBB	A string of six hexadecimal characters, the first two the red component of the color, the next two the green component and the last two the blue component.
RRGGBBAA	A string of eight hexadecimal characters, the first two the red component of the color, the next two the green component, the following two the blue component and the last two the alpha channel.
Date part	Only the date part of the xs:dateTime value should be used.
Time part	Only the time part of the xs:dateTime value should be used.
Ordinal	Each enumeration value may in the implementation correspond to an ordinal number.
Thunk	The value is thunked down to lower precision in the canonical way.
xs:boolean	Conversion to a string should follow the rules specified for the XML datatype xs:boolean.
xs:dateTime	Conversion to a string should follow the rules specified for the XML datatype xs:dateTime.
xs:duration	Conversion to a string should follow the rules specified for the XML datatype xs:duration.

Note: the namespace prefix **xs** is here supposed to be linked with the XML Schema namespace http://www.w3.org/2001/XMLSchema.

5.2.3 Harmonization with XEP-0323 (Sensor Data)

When representing control parameters as momentary field values, it is important to note the similarities and differences between XEP-0323 (Sensor Data) and XEP-0325 (this document):
The **enum** field value data type is not available in XEP-0325 (this document). Instead enumeration valued parameters are represented as **string** control parameters, while the control form explicitly lists available options for the parameter. Options are not available in XEP-0323, since it would not be practical to list all options every time the corresponding parameter was read out. Instead, the **enum** element contains a data type attribute, that can be used to identify the type of the enumeration.
The **numeric** field value data type is not available in XEP-0325 (this document). The reason is that a controller is not assumed to understand unit conversion. Any floating-point valued control parameters are represented by **double** control parameters, which lack a unit attribute. They are assumed to have the same unit as the corresponding **numeric** field value. On the other hand, floating point valued control parameters without units, are reported using the **numeric** field element, but leaving the unit blank.
Control pameters of type **color** have no corresponding field value data type. The color value must be represented in another way, and is implementation specific. Possibilities include representing the color as a string, using a specific pattern (for instance RRGGBBAA), or report

it using multiple fields, one for each component for instance.
The **boolean**, **date**, **dateTime**, **duration**, **int**, **long**, **string** and **time** field value data types correspond to control parameters having the same types and same element names.

5.3 Difference between node parameters and node control parameters

A node defined in a concentrator, as defined by Internet of Things - Concentrators, supporting control has two sets of parameters that are different: First a set of node parameters and then a set of control parameters.
Node parameters are defined by the node type in the concentrator, as described in Internet of Things - Concentrators, and they are typically used by the concentrator to define the node and how to communicate or interact with the underlying device. The important part here is to know that the node parameters are maintained by the concentrator, not the underlying device.
Control parameters however, are parameters that reside on the underlying device. When set, they change the actual state or behaviour of the underlying device. The connection to the device however, controlled by the concentrator, remains unchanged by such a control parameter update.

5.4 Grouping control parameters

Many control actions available in a device can be controlled using only one control parameter. If a device only publishes such control parameters, the order of control parameters is not that important.
However, there are many control actions that require the client to set multiple control parameters at the same time, for the device to have a complete understanding what the client wants to do.
XEP-0141 defines a way to group parameters in a data form by including the concept of pages and sections. Even though these pages and sections are used for layout purposes, it should be used by devices to mark parameters that should be used together to perform control actions.
The following set of rules should be adhered to, by devices as well as clients, to minimize confusion and resulting errors:

- Control parameters should be listed in control forms in the order the device expects the client to write them back.

- Clients should set control parameters in the order they are listed in the corresponding control forms.

- Control actions that require multiple control parameters should report these together, grouped by pages or sections within pages, to make clear that the parameters belong

together.

- For control actions requiring multiple control parameters, devices should strive to publish default values for all parameters involved. These default values should then be used by the device if a client happens to write only a subset of the control parameters required for a control action. The default value could be the current state of the parameter.

Note however, that one cannot always make the assumption that parameters on the same page or same section in a control form belong to the same control action. For instance, a PLC with 16 digital outputs might publish a control form containing a single page with 16 check boxes on (boolean parameters), that can be controlled individually.
To solve the problem of grouping parameters together, so a client can know which parameters belong together, a new element is defined that can be used in data forms: **parameterGroup**. It is optional, but can be added to control parameters in forms, as a way to tell the client that parameters having the same **parameterGroup** belong together and should be written together.
Note: If used, the server must not include a parameter in more than one parameter group at a time. The form may contain multiple group, but each parameter must only have at most one **parameterGroup** element.
The following example illustrates the use of the **parameterGroup** element to group parameters together.

Listing 28: Grouping control parameters

```
<iq type='get'
  from='master@example.org/amr'
  to='spotlight@example.org'
  id='12'>
  <getForm xmlns='urn:xmpp:iot:control' xml:lang='en'/>
</iq>

<iq type='result'
  from='spotlight@example.org'
  to='master@example.org/amr'
  id='12'>
  <x type='form'
    xmlns='jabber:x:data'
    xmlns:xdv='http://jabber.org/protocol/xdata-validate'
    xmlns:xdl:='http://jabber.org/protocol/xdata-layout'
    xmlns:xdd:='urn:xmpp:xdata:dynamic'>
    <title>Spotlight</title>
    <xdl:page label='Output'>
      <xdl:fieldref var='MainSwitch'/>
    </xdl:page>
    <xdl:page label='Direction'>
```

```
      <xdl:fieldref var='HorizontalAngle'/>
      <xdl:fieldref var='ElevationAngle'/>
    </xdl:page>
    <field var='xdd␣session' type='hidden'>
      <value>325ED0F3-9A9A-45A4-9634-4E0D41C5EA06</value>
    </field>
    <field var='MainSwitch' type='boolean' label='Main␣switch'>
      <desc>If the spotlight is turned on or off.</desc>
      <value>true</value>
      <xdd:notSame/>
    </field>
    <field var='HorizontalAngle' type='text-single' label='Horizontal␣
        angle:'>
      <desc>Horizontal angle of the spotlight.</desc>
      <value>0</value>
      <xdv:validate datatype='xs:double'>
        <xdv:range min='-180' max='180'/>
      </xdv:validate>
      <xdd:notSame/>
      <parameterGroup xmlns='urn:xmpp:iot:control' name='direction'/>
    </field>
    <field var='ElevationAngle' type='text-single' label='Elevation␣
        angle:'>
      <desc>Elevation angle of the spotlight.</desc>
      <value>0</value>
      <xdv:validate datatype='xs:double'>
        <xdv:range min='-90' max='90'/>
      </xdv:validate>
      <xdd:notSame/>
      <parameterGroup xmlns='urn:xmpp:iot:control' name='direction'/>
    </field>
  </x>
</iq>
```

The above example informs the client that the two parameters HorizontalAngle and ElevationAngle should be written together to control a control action (named direction).
For more information about common control actions and their parameters, see xep-0000-IoT-Interoperability.html, which defines a set of interoperable interfaces and their abilities.

5.5 Node commands vs. control parameters

Nodes behind a concentrator, as defined in Internet of Things - Concentrators, have an additional means of publishing control interfaces: Node Commands.
However, there are many differences between Node Commands and Control Parameters, as shown in the following list:

- Node Commands are defined by the node type in the concentrator, and not by the

device itself.

- Node Commands may do many different things, not only performing control actions.
- Parametrized Node Commands require the client to always get a parameter data form, and write back values. There's no way to send simple control messages using Node Commands.
- Node Commands can be partitioned, grouped and sorted separately.
- Each Parametrized Node Command has a separate parameter form, which makes grouping of parameters normal.
- Node Commands are only available for nodes controlled by a concentrator.

If implementing a device with many complex control actions (like an advanced PLC), consideration should be made to divide the device into logical groups and implement the concentrators interface as well. Then the more complex control actions could be implemented as Node Commands instead of control actions as defined in this document, and implementing the simpler more intuitive control actions as described in this document.

5.6 Tokens

If control interaction is performed in a context of delegated trust, as defined in the Sensor Network Provisioning XEP-0324 Internet of Things - Provisioning (XEP-0324) [8], tokens should always be included in all calls. This to allow devices to check privileges with provisioning servers.
The **set** and **getForm** commands support the following token attributes:

- **serviceToken**
- **deviceToken**
- **userToken**

For more information about provisioning, see Internet of Things - Provisioning.

[8] XEP-0324: Internet of Things - Provisioning <https://xmpp.org/extensions/xep-0324.html>.

5.7 Controlling devices in large subsystems

Most examples in this document have been simplified examples where a few devices containing a few control parameters have been used. However, in many cases large subsystems with very many actuators containing many different control actions have to be controlled, as is documented in Internet of Things - Concentrators. In such cases, a node may have to be specified using two or perhaps even three ID's: a **sourceId** identifying the data source controlling the device, a possible **cacheType** narrowing down the search to a specific kind of node, and the common **nodeId**. For more information about this, see Internet of Things - Concentrators.
Note: For cases where the **nodeId** is sufficient to uniquely identify the node, it is sufficient to provide this attribute in the request. If there is ambiguity in the request, the receptor must treat the request as a request with a set of nodes, all with the corresponding **nodeId** as requested.

6 Internationalization Considerations

6.1 Time Zones

All timestamps and dateTime values use the XML data type xs:dateTime to specify values. These values include a date, an optional time and an optional time zone.
Note: If time zone is not available, it is supposed to be undefined. The client reading the sensor that reports fields without time zone information should assume the sensor has the same time zone as the client, if not explicitly configured otherwise on the client side.
If devices report time zone, this information should be propagated throughout the system. Otherwise, comparing timestamps from different time zones will be impossible.

6.2 xml:lang

Control commands sent using IQ stanzas instead of messages, should consider using the **xml:lang** attribute to specify the desired language used (if possible) when returning information back to the caller, like error messages, localized control forms, etc.

7 Security Considerations

Controlling devices in a large sensor network is a hackers wet dream. Therefore, consideration of how network security is implemented should not be underestimated. The following sections provide some general items that should be considered.

7.1 Encryption & Authentication

Consider to always use an encrypted connection with any XMPP Server used in the network. Also, make sure the server is properly authenticated and any server certificate properly validated.
Control commands should only be accepted by trusted parties. A minimum is to make sure only authenticated and validated clients (friends) can perform control actions on the device.

7.2 Provisioning

Consider using provisioning servers to allow for detailed control of who can do what in a sensor network. Implementing proper provisioning support decreases the risk for adverse effects, not only from intentional hacking, but also from unintentional errors.
If using delegated trust, make sure the provisioning servers are properly authenticated and validated before trusting them.
More information about provisioning can be found in Internet of Things - Provisioning.

8 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [9].

9 XMPP Registrar Considerations

The protocol schema needs to be added to the list of XMPP protocol schemas.

10 XML Schema

```
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema
  xmlns:xs='http://www.w3.org/2001/XMLSchema'
  targetNamespace='urn:xmpp:iot:control'
  xmlns='urn:xmpp:iot:control'
  xmlns:sn='urn:xmpp:iot:sensordata'
  xmlns:xd="jabber:x:data"
  xmlns:xdv="http://jabber.org/protocol/xdata-validate"
```

[9] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

```
xmlns:xdl="http://jabber.org/protocol/xdata-layout"
elementFormDefault='qualified'>

<xs:import namespace='urn:xmpp:iot:sensordata'/>
<xs:import namespace='jabber:x:data'/>
<xs:import namespace='http://jabber.org/protocol/xdata-validate'/>
<xs:import namespace='http://jabber.org/protocol/xdata-layout'/>

<xs:element name='set'>
  <xs:complexType>
    <xs:choice minOccurs='0' maxOccurs='unbounded'>
      <xs:element name='node' type='NodeReference'/>
      <xs:element name='boolean' type='BooleanParameter'/>
      <xs:element name='color' type='ColorParameter'/>
      <xs:element name='date' type='DateParameter'/>
      <xs:element name='dateTime' type='DateTimeParameter'/>
      <xs:element name='double' type='DoubleParameter'/>
      <xs:element name='duration' type='DurationParameter'/>
      <xs:element name='int' type='IntParameter'/>
      <xs:element name='long' type='LongParameter'/>
      <xs:element name='string' type='StringParameter'/>
      <xs:element name='time' type='TimeParameter'/>
      <xs:element ref='xd:x'/>
    </xs:choice>
    <xs:attributeGroup ref='tokens'/>
  </xs:complexType>
</xs:element>

<xs:element name='setResponse'>
  <xs:complexType>
    <xs:choice minOccurs='0' maxOccurs='unbounded'>
      <xs:element name='node' type='NodeReference'/>
      <xs:element name='parameter' type='Parameter'/>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:element name='paramError' type='ParameterError'/>

<xs:element name='getForm'>
  <xs:complexType>
    <xs:sequence minOccurs='0' maxOccurs='unbounded'>
      <xs:element name='node' type='NodeReference'/>
    </xs:sequence>
    <xs:attributeGroup ref='tokens'/>
  </xs:complexType>
</xs:element>

<xs:element name='parameterGroup'>
```

```
    <xs:complexType>
      <xs:attribute name='name' type='xs:string' use='required'/>
    </xs:complexType>
  </xs:element>

  <xs:attributeGroup name='nodeReference'>
    <xs:attribute name='nodeId' type='xs:string' use='required'/>
    <xs:attribute name='sourceId' type='xs:string' use='optional'/>
    <xs:attribute name='cacheType' type='xs:string' use='optional'/>
  </xs:attributeGroup>

  <xs:attributeGroup name='tokens'>
    <xs:attribute name='serviceToken' type='xs:string' use='optional'/
        >
    <xs:attribute name='deviceToken' type='xs:string' use='optional'/>
    <xs:attribute name='userToken' type='xs:string' use='optional'/>
  </xs:attributeGroup>

  <xs:complexType name='Parameter'>
    <xs:attribute name='name' type='xs:string' use='required'/>
  </xs:complexType>

  <xs:complexType name='BooleanParameter'>
    <xs:complexContent>
      <xs:extension base='Parameter'>
        <xs:attribute name='value' type='xs:boolean' use='required'/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name='ColorParameter'>
    <xs:complexContent>
      <xs:extension base='Parameter'>
        <xs:attribute name='value' type='Color' use='required'/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name='DateParameter'>
    <xs:complexContent>
      <xs:extension base='Parameter'>
        <xs:attribute name='value' type='xs:date' use='required'/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name='DateTimeParameter'>
    <xs:complexContent>
      <xs:extension base='Parameter'>
```

```
      <xs:attribute name='value' type='xs:dateTime' use='required'/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name='DoubleParameter'>
  <xs:complexContent>
    <xs:extension base='Parameter'>
      <xs:attribute name='value' type='xs:double' use='required'/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name='DurationParameter'>
  <xs:complexContent>
    <xs:extension base='Parameter'>
      <xs:attribute name='value' type='xs:duration' use='required'/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name='IntParameter'>
  <xs:complexContent>
    <xs:extension base='Parameter'>
      <xs:attribute name='value' type='xs:int' use='required'/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name='LongParameter'>
  <xs:complexContent>
    <xs:extension base='Parameter'>
      <xs:attribute name='value' type='xs:long' use='required'/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name='StringParameter'>
  <xs:complexContent>
    <xs:extension base='Parameter'>
      <xs:attribute name='value' type='xs:string' use='required'/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name='TimeParameter'>
  <xs:complexContent>
    <xs:extension base='Parameter'>
      <xs:attribute name='value' type='xs:time' use='required'/>
```

```
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:complexType name='ParameterError'>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute name='var' type='xs:string' use='required'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name='NodeReference'>
    <xs:attributeGroup ref='nodeReference'/>
  </xs:complexType>

  <xs:simpleType name='Color'>
    <xs:restriction base='xs:string'>
      <xs:pattern value='^([0-9a-fA-F]{6})|([0-9a-fA-F]{8})$'/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

11 For more information

For more information, please see the following resources:

- The Sensor Network section of the XMPP Wiki contains further information about the use of the sensor network XEPs, links to implementations, discussions, etc.
- The XEP's and related projects are also available on github, thanks to Joachim Lindborg.
- A presentation giving an overview of all extensions related to Internet of Things can be found here: http://prezi.com/esosntqhewhs/iot-xmpp/.

12 Acknowledgements

Thanks to Joachim Lindborg and Tina Beckman for all valuable feedback.