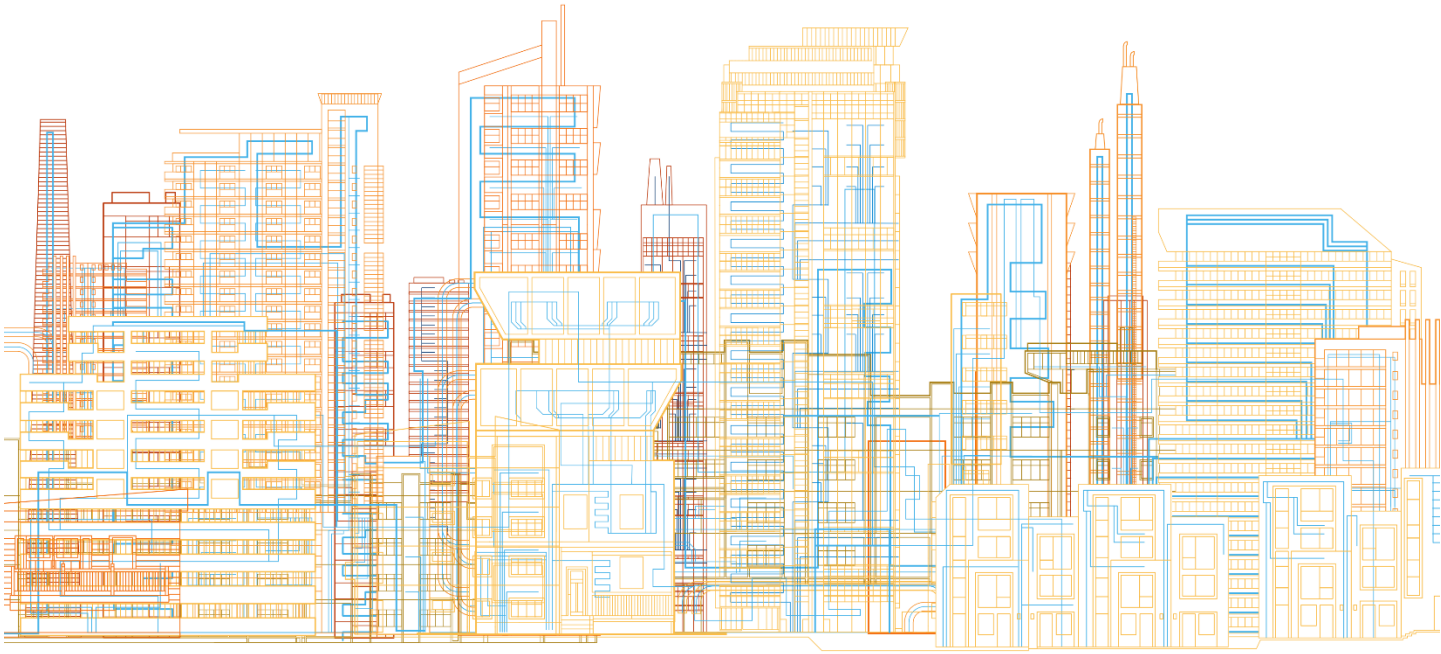


Specification

Controlling Service Point Level Smart Devices in a Distribution Network using IEC CIM

sgem
Smart Grids and Energy Markets

DATE	AUTHOR	COMPANY	CHANGES
12.8.2014	Risto-Matti Keski-Keturi	ABB	Initial version

Contents

1. ABOUT THIS DOCUMENT

This document contains specifications for developers to implement IEC 61968-9 and IEC 61968-100 and integrate devices under a distribution network connection point into a larger smart grid entity.

The described interface is intended and suitable for managing large variety of devices within the Smart Grid taking into consideration the known and foreseen technical and market requirements. It covers data acquisition and control of home and industrial automation, demand response and various distributed energy resources. This document is accompanied by WSDL and XML Schema documents to formally describe the implementation according to the specification.

1.1. Background

During the Finnish Smart Grids and Energy Markets research program (SGEM 2010-2014), the potential and need was identified to utilize the increasing number of smart devices, owned by distribution network customers, for demand response and demand management. At the same time, these same smart devices were additionally seen as a great pathway to improved and extended offering of other “customer services”. The existing service providers such as distribution network operators and electricity vendors and foreseen new service providers may in addition to traditional and demand response services offer a versatile variety home, premise, energy and safety and other type of services for the consumers and prosumers of distribution networks.

The working group behind this document identified the parallels of these two goals and began to look for the best way to converge them. A major obstacle to realizing the potential benefits was the lack of a commonly agreed way of combining this fragmented data into information. After a thorough search and study of several alternative candidates (e.g. OpenADR), the IEC CIM series of standards was chosen. The IEC CIM covers modeling of distribution network hierarchy, structure, elements and devices and additionally contains the necessary methods and flexibility to support the future variety within the modern smart society.

The interface described here should be understood as a specification of the desired set of functionality for enabling the aforementioned goals. The referenced standards should be used as the primary specification, while this document is a pointer to the relevant standards.

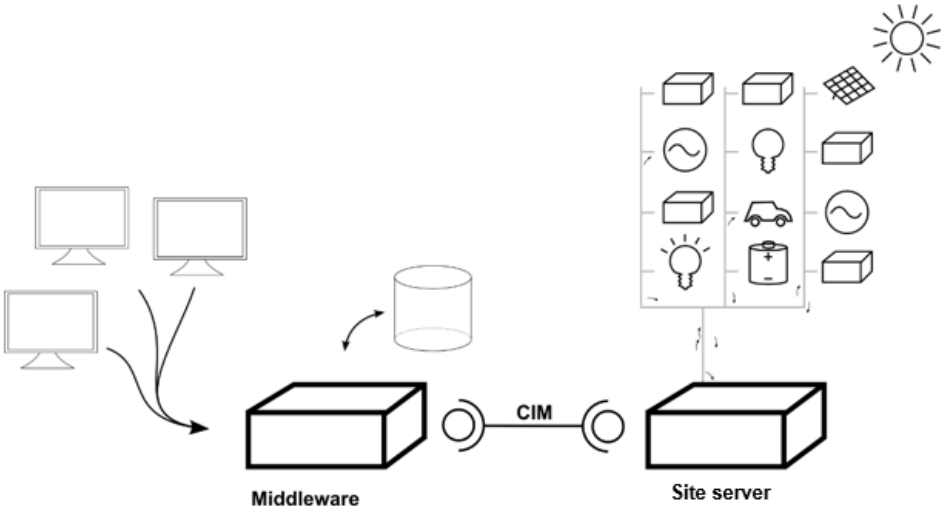
1.2. References

Published CIM standards related to this document are IEC 61968-1: Interface architecture and general requirements, edition 2.0 and IEC 61968-100: Implementation Profiles for IEC 61968. IEC 61968-100 is used for defining the message containers and IEC 61968-9 for the specific messages.

The UML diagrams within this document were created using version 12 v 08 of the Enterprise Architect model from the CIM User Group, obtained from

http://cimug.ucaiug.org. A draft version of this document was released internally within the working group in spring 2014 before the pilot case implementation. .

2. DESCRIPTION OF THE SYSTEM



The setup described in this document.

The open interface between two systems is referred from now on as *Site server* and *Middleware*.

The *Site server* is a server that controls multiple peripheral devices, such as electrical loads or energy resources, or a single intelligent device. It communicates using an internal protocol with appliances within and under its control. The Site server shares data with the middleware using the standard interface described in this document.

The middleware is responsible for producing user interfaces for a wide array of devices, relaying data from the interface to a user interface or backend system and possibly storing some configuration or historical data as necessary.

2.1. Coverage

This interface specification in its current form describes the aspects listed below.

- Receiving and identifying measurements from the site server to the middleware, eg. value of current on a line or status or a switch open/close status
- Sending control actions from the middleware to a site server to be executed on the devices
- Receiving notifications in the middleware of events taking place in the system, eg. a pushbutton being pressed or an alarm limit being reached
- Querying the configuration of a system, eg. requesting a list of appliances available for control by the middleware from the site server

Functionality proposed but left out from this interface at this initial stage include:

- Describing control groups, eg a list list of control actions identified by a single control type. Instead, such groups should be defined only within the middleware and not propagated to the site server. In case a functionality is to be mapped to a physical button or similar, the event of pressing the button should be exported to the interface and any mapping to control actions should be maintained only in the middleware.

2.2. Technology

The interface described here will be implemented using XML and described as an XML schema. The outlying transmission technology will be SOAP over HTTP(S).

2.3. Security

The transmission link will use the HTTPS protocol for transmission. The security presumption is that the site server will have one communication partner that it can reliably identify and will have full trust in. This can be achieved by setting a single certificate as trusted and allowing all communication from a counterparty that can produce this predefined certificate.

Implementations may alternatively contain methods of defining multiple certificates and an access control list for parts of the system behind one server, eg. devices 1,2,3 can be controlled from a counterparty identified by certificate A and devices 3 and 4 can be controlled by a counterparty identified by certificate B. This may be useful for example when an apartment building is controlled via one shared server.

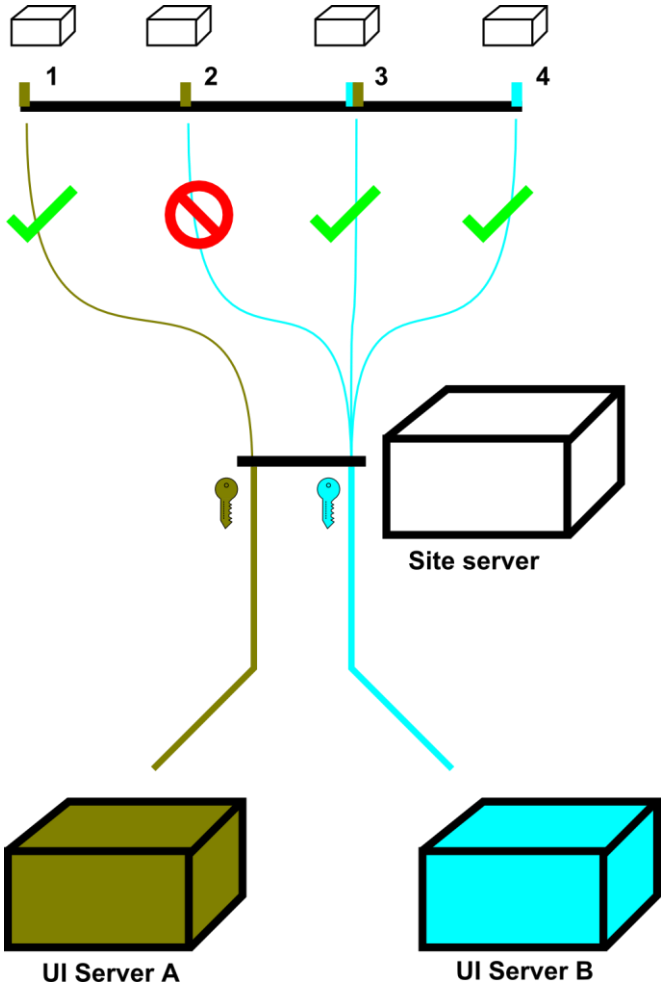


Figure 1 The security model

In this model, managing user credentials (e.g. password changes, additional identification methods etc.) does not have to be done on the site server, which is meant to be usable with minimum maintenance load. On this server, only the allowable service providers are configured.

In an installation where the devices under a single server are owned by multiple independent parties (e.g. in an apartment building), the procedure for a device owner to enable the interface would thus be

1. Make contract with a service provider
2. Deliver details of the certificate of the provider to the organization responsible for the site server
    a. Organization responsible for site server will add mapping from key of the organization to the device in question in the certificate list of the site server
3. Service provider will look up publicly available communication details of the site server

3. CIM DATA TYPES

The basic concept of this interface is that there are multiple EndDevices in a hierarchy representing controllable loads. An EndDevice may be a real device, such as a car charge port, or a virtual device such as a controllable object in a site network.

This allows measurements, controls and descriptions to be allocated to the exact device they relate to. The devices available can be queried from the server, with all the necessary metadata attached that will allow identification of a specific controllable object.

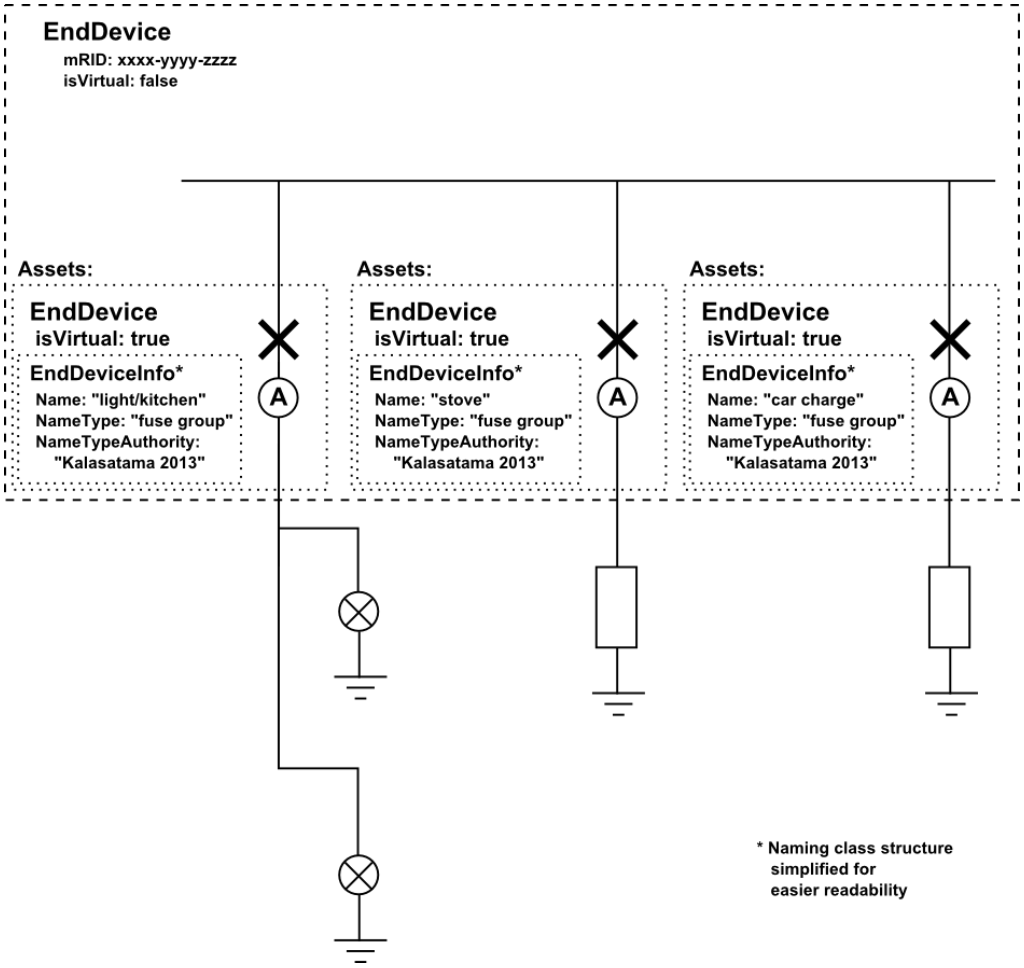


Figure 2 Representing KNX feeders in an apartment as individual EndDevices

Figure 3 shows the classes that describe an EndDevice. While these include many ways to describe the functionality of an EndDevice, ultimately sufficient is if both sides of the interface understand a combination of Name and NameType. The final version of IEC 61968-9 is expected to define names of common types of at least EndDeviceEvents and EndDeviceControls, so in this interface the IdentifiedObject.name field is left empty to allow adding a primary

name from the standard to object already named by this specification. While the existing version of the published standard has not defined names for EndDeviceInfos, this seems like sound practice and in line with CIM modeling, which is needed to solve a specific problem in this implementation.

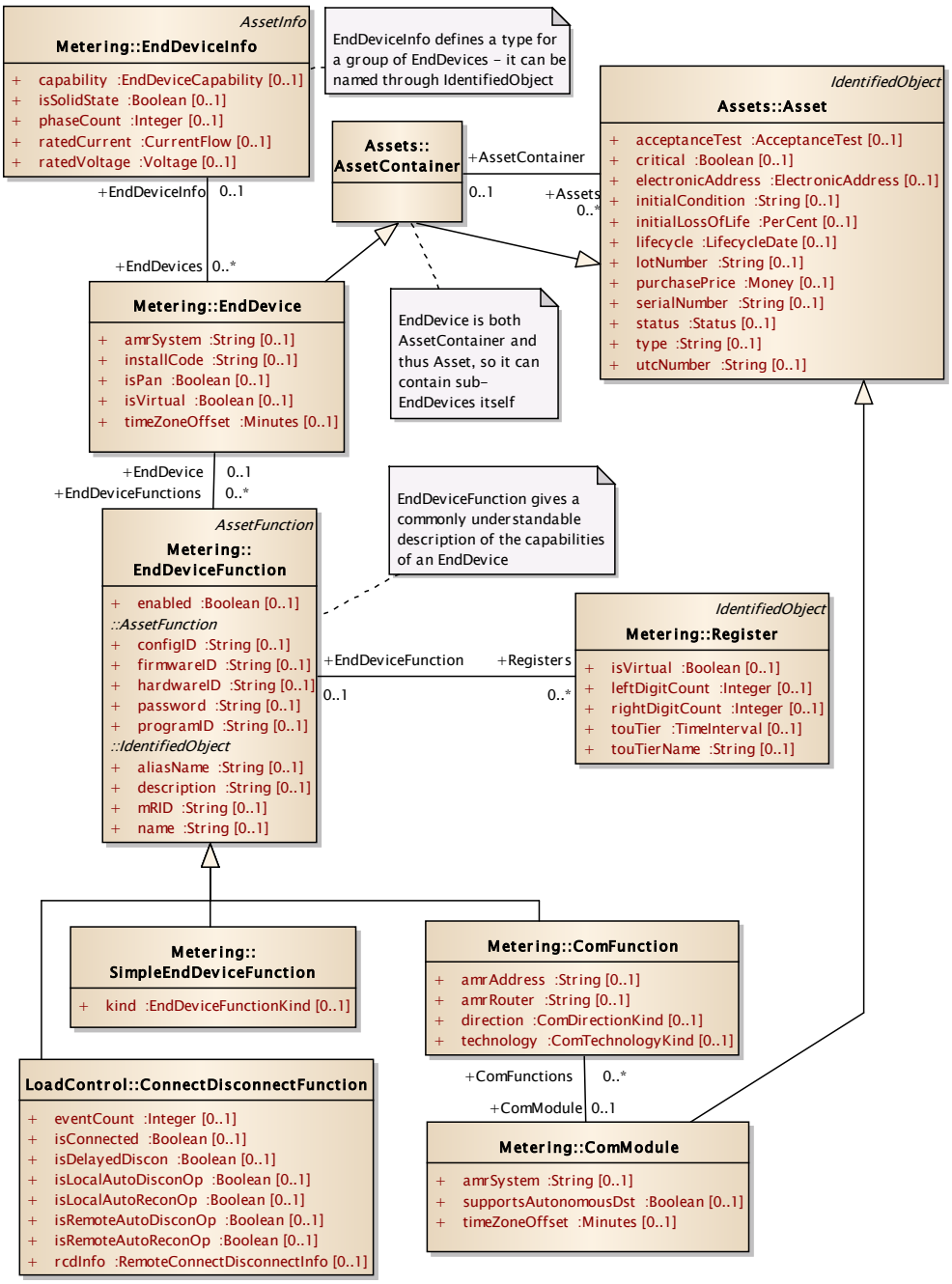


Figure 3 The classes that describe an EndDevice

An example of mapping actual functionality of the feeder group to the CIM description classes is shown below.

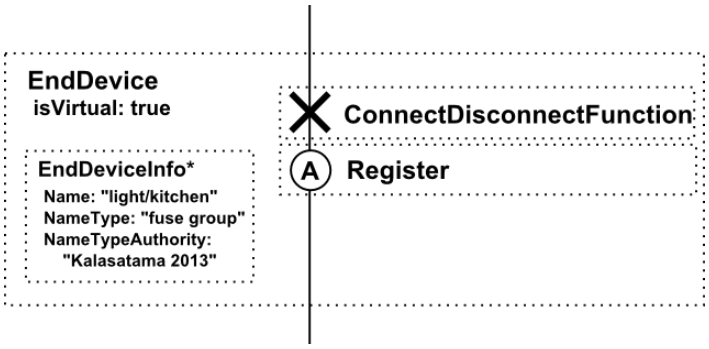


Figure 4 Describing a single fuse group with CIM objects

Here additional classes, ConnectDisconnectFunction and Register are linked to the EndDevice. From these, the other endpoint may deduce the available capabilities of the device.

There currently seems to be no clear way in the CIM to describe which EndDeviceControlTypes are enabled by which EndDeviceFunction. This may be a deficiency in the CIM model or an oversight by the authors of this document.

3.1. Names in CIM

Finally, figure 4 shows the classes used for naming objects in the CIM. The identifiers in IdentifiedObject, *mRID* and *name* should be used to uniquely identify an object, mRID being a globally unique identifier and name being a human-readable code which at least in some cases may be defied by further releases of the standard. While theoretically every IdentifiedObject has an mRID, the CIM does not require it to be defined where it is not needed. In the case of this interface, each virtual feeder EndDevice does indeed exists as an individual object (eg. it has serial number), but the interface does not have to use this identifier when referring to the device.

The Name class in turn can be used to assign an arbitrary number of names to any object. The type of name can be specified, so that eg. a name defined in the standard can be distinguished from a commonly agreed name defined in this document. A number of can be assigned to any CIM object.

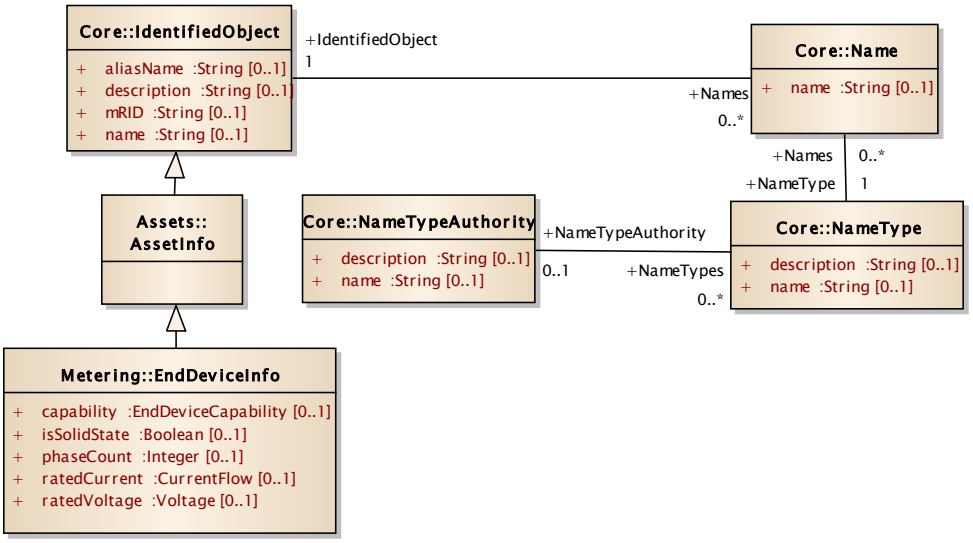


Figure 5 Classes that identify the type of an EndDevice. Each EndDevice links to a single EndDeviceInfo, which describes it's type but each EndDeviceInfo can link to multiple EndDevices of the same type. Similar naming can apply to any CIM object.

The XML snippet below shows how to refer to a specific virtual EndDevice in a control message without knowing the exact identifier of it. Written as text, the snippet below would mean "the 'stove' type feeder under the KNX box called aaa-bbb-ccc".

```
<EndDevice>
  <!-- here could be an mRID field, but in this case we
       do not care about the exact 'serial number' of this
       specific feeder -->
  <EndDeviceInfo>
    <Names>
      <!-- this tells us we refer to and abstract
           feeder type 'stove' as defined by the
           Kalasatama Consortium, eg. this document -->
      <name>stove</name>
      <NameType>
        <name>Kalasatama abstract types 2013</name>
        <NameTypeAuthority>
          <name>Kalasatama Consortium</name>
        </NameTypeAuthority>
      </NameType>
    </Names>
  <EndDeviceInfo>
  <AssetContainer xsi:type="EndDevice">
    <!-- this is the unique identifier of the KNX control
         box of the apartment, which contains multiple
         sub-EndDevices -->
    <mRID>aaa-bbb-ccc-ddd</mRID>
  </AssetContainer>
<EndDevice>
```

3.2. Controls and events

A control command to an EndDevice uses the class EndDeviceControl. The type of the control is identified by EndDeviceControlType, which includes classification of the control and through IdentifiedObject a Name as defined in this document.

Some controls, such as “open load break switch” will most likely have a direct mapping with a type defined in IEC 61968-9 in the future, while with others a direct mapping may not be available, eg “set living room light to blink in green color”.

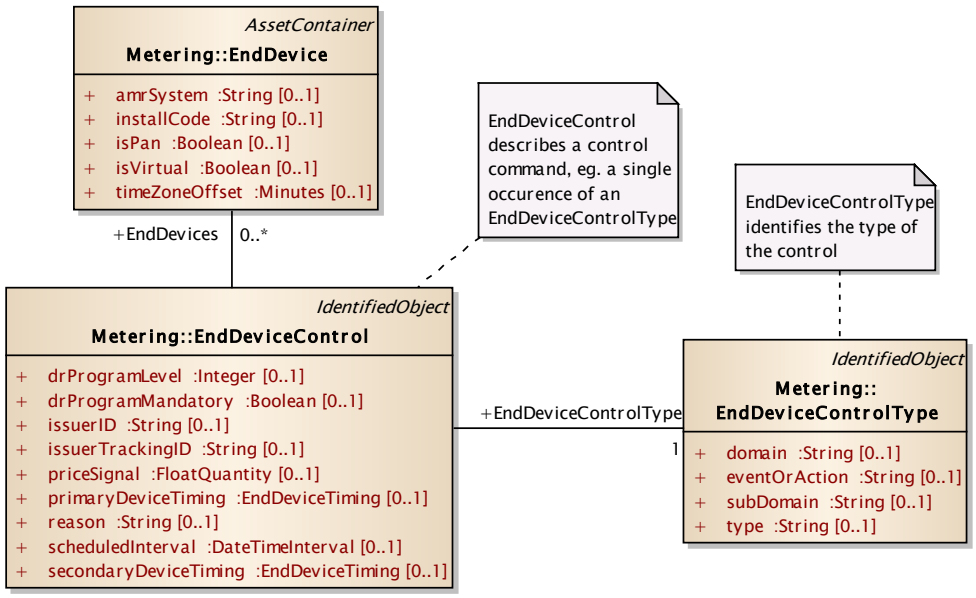


Figure 6 EndDeviceControl, a command to the EndDevice to perform a function

The class EndDeviceEvent can be used to communicate something happening in the device. An event may be generated when a pushbutton is pressed, an alarm limit of a measurement is exceeded etc.

Here again EndDeviceEventType describes what the event means while EndDeviceEvent describes the instance of an EndDeviceEventType. Additional data in the form of key-value pairs can be attached to the event using the class EndDeviceEventDetail.

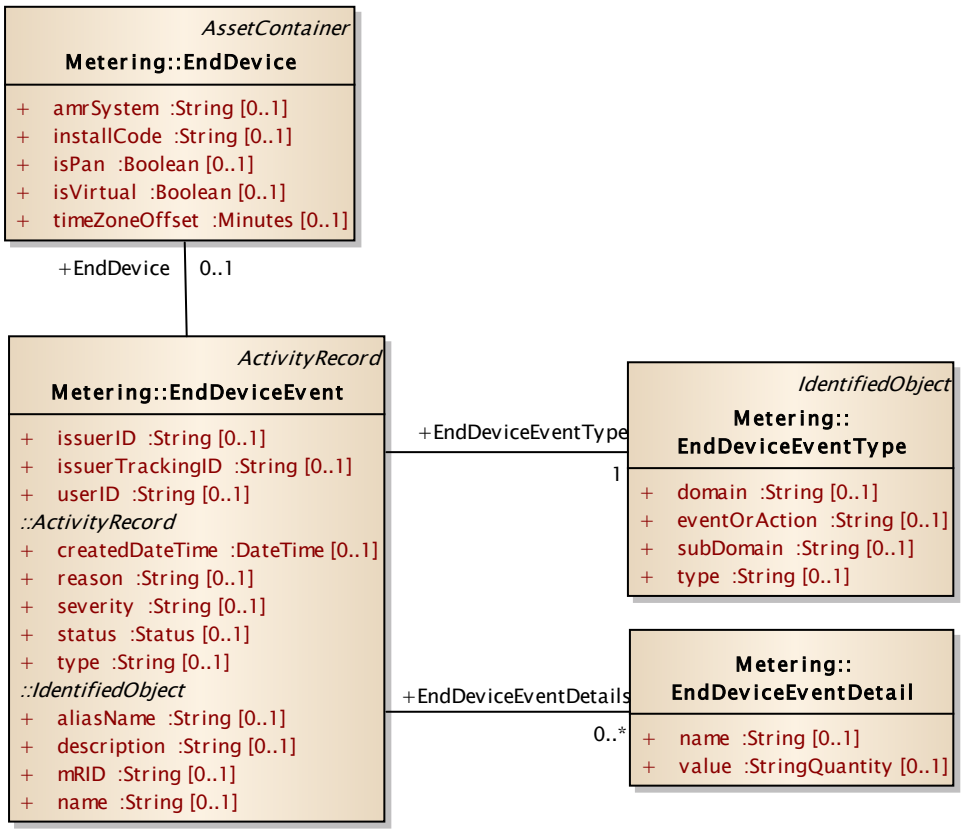


Figure 7 EndDeviceEvent, a class that signals the middleware when something happens.

Finally, measurements of both numeric and discrete quantities can be transferred using the class MeterReading. With this class, type of the reading is transferred in ReadingType which is analogous to EndDeviceEventType and EndDeviceControlType.

Note that an EndDeviceEvent may also include readings if desired, as an example an alarm may include the value of the measured quatity as a linked MeterReading.

As MeterReading only links to Meter and not to EndDevice, any EndDevice that is capable of producing measurements, be it number or on/off information, must be of type Meter. This is a cosmetic detail as Meter is a subclass of EndDevice.

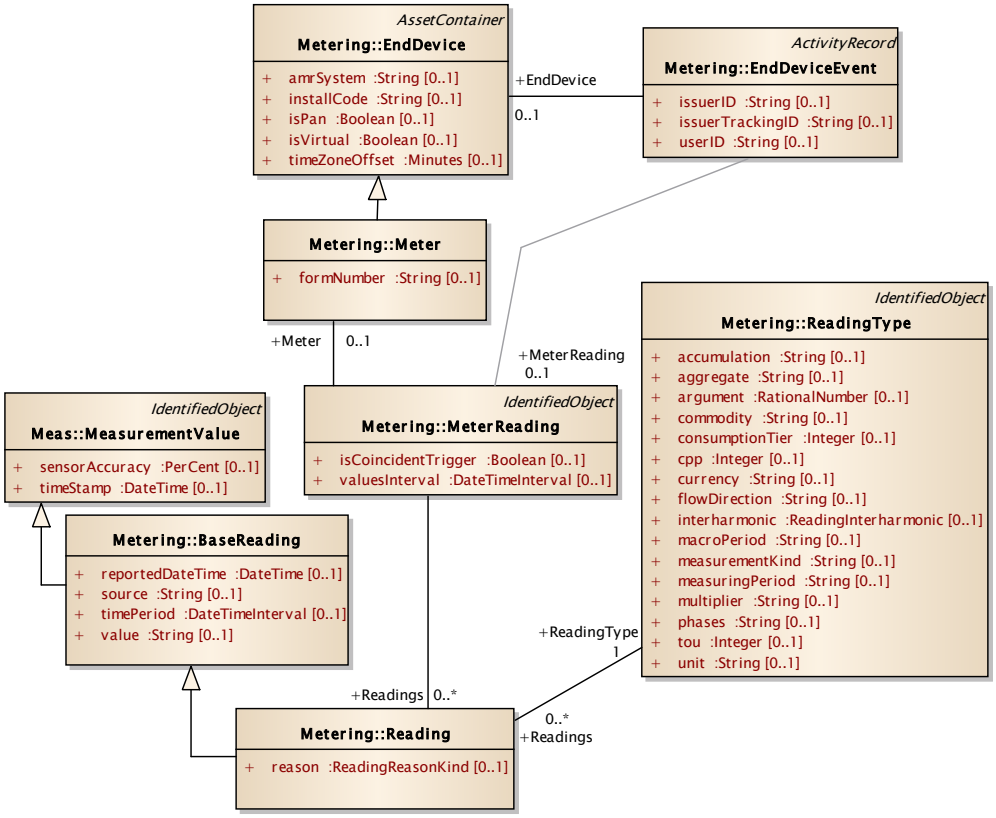


Figure 8 MeterReading can transfer measurements

3.3. Values used for specific fields

3.3.1. EndDevice

isVirtual: shall be true when describing a 'feed' type EndDevice and false for actual devices. Can be omitted from requests.

3.3.2. EndDeviceControl

issuerID: shall be the name of the organisation sending the message, eg. 'Helsingin Energia'

reason: shall be "UI" when the control is the direct result of user action in the interface. If the control is due to a programmed action, eg. automatically generated but based on a rule by the user, shall be "programmed". If the control is due to a demand response agreement, reason shall be "DR".

drProgramMandatory: shall be true if the customer has a contractual obligation to implement the requested control and false if not (eg. the control will result in a benefit but no penalty). This field shall only be defined for controls originating from a demand response agreement.

3.4. Standard types supported

When an EndDeviceEventType, EndDeviceControlType or ReadingType is defined in IEC 61968-9, it should be referred to according to the XML snippet below.

```
<EndDeviceEventType ref="x.y.z"/>
```

This section defines the IEC standard codes that can be used within this interface. See below for a usage example.

First, the codes for EndDeviceEventType. See IEC 61968-9 ed 2 Annex E for more information.

15.26.83.289	A local control has changed state. Additional information must be included in the EndDeviceEventDetails structure, see below. Only a subset of local controls may be exposed via this interface, depending on the installation.

For EndDeviceControlType. See IEC 61968-9 ed 2 Annex F for more information.

12.31.0.23	Open Remote Connect/Disconnect Switch
12.31.0.18	Close Remote Connect/Disconnect Switch
58.31.0.23	Stop charging car
58.31.0.18	Begin charging car
2.15.26.30	Change demand response settings. A proper PanDemandResponse object must be attached to such EndDeviceControl to identify the new duty cycle, load adjustment of heating/cooling setpoint.

And finally ReadingType. Note that spaces should not be included in the type, they are here to ease readability. See IEC 61968-9 ed 2 Annex C for more information.

0.0.0.6.0.1 .4 .0.0.0.0.0.0.0.0.0.5 .0	Current in Amperes
0.0.0.6.0.1 .54 .0.0.0.0.0.0.0.0.0.29 .0	Voltage in Volts
0.0.0.6.1.1 .37 .0.0.0.0.0.0.0.0.0.38 .0	Power in Watts
0.0.0.6.0.9 .155.0.0.0.0.0.0.0.0.0.137.0	Water flow in liters per hour

0.0.0.6.0.12.37 .0.0.0.0.0.0.0.0.0.38 .0	Heating water energy flow in Watts.
0.0.0.6.0.4 .46 .0.0.0.0.0.0.0.0.0.23 .0	Room temperature in degrees Celsius.
0.0.7.9.1.1 .12 .0.0.0.0.0.0.0.0.0.72 .0	Energy consumption in Watt-hours in the past hour.
0.0.7.9.0.9 .58 .0.0.0.0.0.0.0.0.0.134.0	Water consumption in liters in the past hour.
0.0.0.0.0.0 .140.0.0.0.0.0.0.0.0.0.109.0	Switch status, with values in "1" for switch closed (energized) and "0" for switch open (de-energized)

3.5. Event details

Additional information about events must be included in the EndDeviceEventDetails structure when the event type is one of the listed.

15.26.83.289: This event identifies that a user-accessible switch changed position. The switch and new position must be identified with details named "state" and "switch" according to the table below.

switch	**state**	**description**
switch	on	Sent when a normal switch has been set to the "on" state, eg a device is energized.
	off	Sent when a normal switch has been set to the "off" state.
home-away-switch	home	Sent when the home-away switch has been set to the *home* position.
	away	Sent when the home-away switch has been set to the *away* position.
dimmer-switch	0…100	A dot separated decimal string between 1 and 100 indicating the set point of a dimmer type switch.

3.6. Identifiers

The interface must identify many different types of information: usage points, control types, measurement value sources etc. This section defines the NameTypeAuthorities and NameTypes that must be used by the implementation. Additional identifiers may be added to enable interoperability with new systems.

The table below lists the types of objects that need to be identified by the names described here.

EndDevice	A class describing a single EndDevice, eg a feeder in the KNX system.

The following NameTypeAuthorities are used

Kalasatama 2013	NameTypes defined under this authority are identifiers that are used in this document.

The NameTypeAuthority “Kalasatama 2013” defines the NameTypes as per the table below.

Group type	This name is attached to an EndDeviceInfo. It can be used to distinguish between such feed groups as “stove” or “car charge”. The allowable names are defined below. This name does not have to be unique, eg one EndDeviceInfo may property phaseCount of ‘3’, another of ‘1’ and yet they can share the same ‘Group type’ name. Only one name of this type can be attached to a single EndDeviceInfo.
Room type	For a group type that may have multiple instances within an apartment (eg. ‘ligth’), a Name with this type may be added to distinguish between separate installations of that same type within the site.

In addition, the following NameTypes shall be used. The NameTypeAuthority used with these NameTypes shall be a string that is unique within one distribution system operator.

Usage point code	This `Name` identifies a `UsagePoint` in a manner that is unique within one `NameTypeAuthority`, eg. a DSO. The site server may assume that all requests refer to only one DSO code. It does not need to check the `NameTypeAuthority.name`.
	
	`<Names>`
	`  <name>1234567</name>`
	`  <NameType>`
	`  <name>Usage point code</name>`
	`  <NameTypeAuthority>`
	`  <name>DSO LLC LTD OY AB</name>`
	`  </NameTypeAuthority>`
	`  </NameType>`
	`</Names>`

A Name of type “Group type” can have values according to the list below. The intention is to describe the most common types here to minimize engineering overhead. When a grouping does not fit the categories defined here, an EndDeviceInfo class cannot be used and instead the EndDevice must be named with a “User Identifier” as specified above.

- light
- stove
- air conditioning
- refridgerator
- plugs
- laundry
- dishwasher
- car charge
- car heat
- sauna heat
- floor heat
- direct heating
- water heating
- energy storage

- other

A Name of type “Room type” must have a value according to the list below. The same Group/Room pair may be defined more than once, in which case there may be an additional name or the user must remember which mRID is which room. If no applicable name can be found for the specific installation (eg. an independent distributed energy resource site), this name may be omitted.

- kitchen
- bedroom
- livingroom
- bathroom
- utility room
- warehouse
- public space
- corridor

3.7. Methods in the interface

This section describes the methods in the interface. The names of these are decided based on the guidelines laid out in IEC 61968-100 annex D, Generic WSDL.

The interface definition is distributed in three parts. Directory “Full” contains the profiles defined in IEC 61968-9 edition 2 and the generic web service of IEC 61968-100. Both parties of the exchange must implement this interface.

Directories “Site” and “External” contain a subset of the profiles in the standard that is defined in this document. These files are provided for the convenience of the implementer. The restricted files may be used to generate server and client code, as long as an error handler is added that can provide standard-compliant replies to clients that try to connect with the profiles specified in the standard.

All messages that satisfy the schemas in “Site” and “External” will also satisfy the standard schemas. All messages that satisfy the “Full” schema may not satisfy “Site” or “External” schemas.

The IEC 61968-9 profiles (message schemas) used in this specification are

- MeterReadings
- MeterReadSchedule
- GetMeterReadings
- EndDeviceEvents
- EndDeviceControls

An additional profile is defined for querying the devices of an apartment, which supports the EndDevice -> sub-EndDevice structure specified above, and linking measurement types to EndDevices.

4. USE CASES AND EXAMPLES

4.1. Querying devices in an apartment

Before issuing commands to specific feeders, the middleware must query the site server for a list of `EndDevices` by using the function `GetEndDevices`.

Code listing 1

```
<Message
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Header xmlns="http://iec.ch/TC57/2011/schema/message">
    <Verb>get</Verb>
  </Header>
  <Request xmlns="http://iec.ch/TC57/2011/schema/message">
    <ID idType="Usage point code"
        idAuthority="DSO Oy Ab">123456</ID>
  </Request>
</Message>
```

The site server shall reply with the main physical `EndDevice` and all sub-`EndDevices` filled in as per the example below.

Code listing 2

```
<?xml version="1.0"?>
<ResponseMessage
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>EndDeviceConfig</Noun>
  </Header>
  <Reply>
    <Result>OK</Result>
    <Error>
      <code>0.0</code>
    </Error>
  </Reply>
  <Payload>
    <EndDeviceConfig
        xmlns="http://iec.ch/TC57/2011/EndDeviceConfig#">
      <EndDevice>
        <mRID>81e62a5f-9955-4502-8477-05b1b6b8d0b4</mRID>
        <isPan>true</isPan>
        <Names>
          <name>light</name>
          <NameType>
            <name>Group type</name>
            <NameTypeAuthority>
              <name>Kalasatama 2013</name>
```

```
            </NameTypeAuthority>
          </NameType>
        </Names>
        <Names>
          <name>bedroom</name>
          <NameType>
            <name>Room type</name>
            <NameTypeAuthority>
              <name>Kalasatama 2013</name>
            </NameTypeAuthority>
          </NameType>
        </Names>
      </EndDevice>
      <EndDevice>
        <mRID>c84d9eed-6388-4221-ad00-17d0f0f0fca2</mRID>
        <isPan>true</isPan>
        <Names>
          <name>floorheat</name>
          <NameType>
            <name>Group type</name>
            <NameTypeAuthority>
              <name>Kalasatama 2013</name>
            </NameTypeAuthority>
          </NameType>
        </Names>
        <Names>
          <name>kitchen</name>
          <NameType>
            <name>Room type</name>
            <NameTypeAuthority>
              <name>Kalasatama 2013</name>
            </NameTypeAuthority>
          </NameType>
        </Names>
      </EndDevice>
    </EndDeviceConfig>
  </Payload>
</ResponseMessage>
```

4.2. On/Off control of a fuse group

The UI Server must be able to command the site server to turn a fuse group switch on or off. The apartment receiving the command must be identified with an absolute identifier and the fuse group with an abstract name, eg. ‘lighting’.

The example below turns off lights in the bedroom. The identifier must be queried before the call by issuing a `GetEndDeviceConfig` command.

Code listing 3

```
<Message
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>create</Verb>
    <Noun>EndDeviceControl</Noun>
  </Header>
```

```
  <Payload>
    <EndDeviceControl
        xmlns="http://iec.ch/TC57/2011/EndDeviceControls#">
      <EndDeviceControlType ref="12.31.0.23" />
      <EndDevices>
        <mRID>81e62a5f-9955-4502-8477-05b1b6b8d0b4</mRID>
      </EndDevices>
    </EndDeviceControl>
  </Payload>
</Message>
```

If the control is part of a demand response agreement the customer is engaged in, the control message should contain information as in the example below.

Code listing 4

```
<Message
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>create</Verb>
    <Noun>EndDeviceControl</Noun>
  </Header>
  <Payload>
    <EndDeviceControl
        xmlns="http://iec.ch/TC57/2011/EndDeviceControls#">
      <drProgramMandatory>true</drProgramMandatory>
      <issuerID>Electricity Retailer &amp;amp; Co Ltd</issuerID>
      <EndDeviceControlType ref="12.31.0.23" />
      <EndDevices>
        <mRID>81e62a5f-9955-4502-8477-05b1b6b8d0b4</mRID>
      </EndDevices>
    </EndDeviceControl>
  </Payload>
</Message>
```

If a demand response control is valid only for a predetermined time, it shall be communicated using the primaryDeviceTiming structure. When the time (in minutes) specified is over, the system shall return to the state that it was before receiving the command, unless an overriding control has been given[1].

Code listing 5

```
<Message
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>create</Verb>
    <Noun>EndDeviceControl</Noun>
  </Header>
  <Payload>
```

[1] If a demand response program controls a heater to turn off and the local heater control is set to on, the heater will be turned off. If within the time of this control the user changes the local control to off, the heater will not be turned on after the period of demand response control is over.

```
    <EndDeviceControl
        xmlns="http://iec.ch/TC57/2011/EndDeviceControls#">
      <drProgramMandatory>true</drProgramMandatory>
      <issuerID>Electricity Retailer &amp;amp; Co Ltd</issuerID>
      <EndDeviceControlType ref="12.31.0.23" />
      <EndDevices>
        <mRID>81e62a5f-9955-4502-8477-05b1b6b8d0b4</mRID>
      </EndDevices>
      <primaryDeviceTiming>
        <duration>120</duration>
        <durationIndefinite>false</durationIndefinite>
        <interval />
      </primaryDeviceTiming>
    </EndDeviceControl>
  </Payload>
</Message>
```

4.3. Subscribing to measurements and receiving updates

To subscribe to measurements, the receiver creates a MeterReadSchedule as below. The recurrencePeriod defines the number of seconds desired between readings. The subscriber may define a filter by listing the desired ReadingTypes. The scheduleInterval defines the date time range that this request is valid. After such period the measurements are no longer delivered. Each MeterReadSchedule created overrides any previous MeterReadSchedule with a longer recurrencePeriod for the duration of the new schedule.

Code listing 6

```
<Message
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>create</Verb>
    <Noun>MeterReadSchedule</Noun>
  </Header>
  <Payload>
    <MeterReadSchedule
        xmlns="http://iec.ch/TC57/2011/MeterReadSchedule#">
      <TimeSchedule>
        <recurrencePeriod>5</recurrencePeriod>
        <scheduleInterval>
          <end>2014-02-26T09:02:01.9529235+02:00</end>
          <start>2014-02-26T08:57:01.9519234+02:00</start>
        </scheduleInterval>
      </TimeSchedule>
      <UsagePoint>
        <Names>
          <name>1234567</name>
          <NameType>
            <name>Usage point code</name>
            <NameTypeAuthority>
              <name>DSO LLC LTD OY AB</name>
            </NameTypeAuthority>
          </NameType>
        </Names>
```

```
      </UsagePoint>
    </MeterReadSchedule>
  </Payload>
</Message>
```

The measurement message sent by the site server is shown below.

Code listing 7

```
<?xml version="1.0" encoding="utf-8" ?>
<Message xmlns="http://iec.ch/TC57/2011/schema/message"
         xmlns="http://iec.ch/TC57/2011/MeterReadings#">
  <Header>
    <Verb>created</Verb>
    <Noun>MeterReadings</Noun>
  </Header>
  <Payload>
    <MeterReadings>
      <MeterReading
           xmlns="http://iec.ch/TC57/2011/MeterReadings#">
        <Meter>
          <mRID>81e62a5f-9955-4502-8477-05b1b6b8d0b4</mRID>
        </Meter>
        <Readings>
          <reason>inquiry</reason>
          <timeStamp>2014-04-14T18:10:21Z</timeStamp>
          <value>231.3</value>
           <!--current in Volts -->
          <ReadingType
           ref="0.0.0.6.0.1.54.0.0.0.0.0.0.0.0.0.29.0"/>
        </Readings>
      </MeterReading>
    </MeterReadings>
  </Payload>
</Message>
```

4.4. Querying stored readings

Depending on the storage capacity at the site, the site server may allow querying past measurements. If multiple readings are available, the time series will be transmitted as one MeterReadings element with multiple Reading elements.

```
<?xml version="1.0" encoding="utf-8" ?>
<Message xmlns="http://iec.ch/TC57/2011/schema/message"
         xmlns="http://iec.ch/TC57/2011/MeterReadings#">
  <Header>
    <Verb>create</Verb>
    <Noun>GetMeterReadings</Noun>
  </Header>
  <Payload>
    <GetMeterReadings>
        <Meter>
          <mRID>81e62a5f-9955-4502-8477-05b1b6b8d0b4</mRID>
        </Meter>
        <!--current in Volts -->
        <ReadingType
           ref="0.0.0.6.0.1.54.0.0.0.0.0.0.0.0.0.29.0"/>
```

```
        <TimeSchedule>
          <scheduleInterval>
            <end>2014-02-26T09:02:00+02:00</end>
            <start>2014-02-26T08:57:00+02:00</start>
          <scheduleInterval>
        <TimeSchedule>
    </GetMeterReadings>
  </Payload>
</Message>
```

4.5. Receiving events

The status of a home-away switch is communicated using the following event:

Code listing 8

```
<Message
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>created</Verb>
    <Noun>EndDeviceEvent</Noun>
  </Header>
  <Payload>
    <EndDeviceEvent
        xmlns="http://iec.ch/TC57/2011/EndDeviceEvents#">
      <createdDateTime>
        2014-02-27T11:43:33.1874521Z
      </createdDateTime>
      <EndDeviceEventDetails>
        <name>state</name>
        <value>home</value>
      </EndDeviceEventDetails>
      <EndDeviceEventDetails>
        <name>switch</name>
        <value>home-away-switch</value>
      </EndDeviceEventDetails>
      <EndDeviceEventType ref="15.26.83.289" />
      <UsagePoint>
        <Names>
          <name>1234567</name>
          <NameType>
            <name>Usage point code</name>
            <NameTypeAuthority>
              <name>DSO LLC LTD OY AB</name>
            </NameTypeAuthority>
          </NameType>
        </Names>
      </UsagePoint>
    </EndDeviceEvent>
  </Payload>
</Message>
```

4.6. General reply

After receiving a message that does not require any data in return, each member of the interface shall reply with an acknowledgement message as below. All error codes defined in IEC 61968-9 may be used.

Code listing 9

```
<ResponseMessage
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>EndDeviceControl</Noun>
  </Header>
  <Reply>
    <Result>OK</Result>
    <Error>
      <code>0.0</code>
    </Error>
  </Reply>
</ResponseMessage>
```