IEC TR 61968-900

Edition 1.0 2015-10

TECHNICAL REPORT

colour inside

Application integration at electric utilities – System interfaces for distribution management –
Part 900: Guidance for implementation of IEC 61968-9

IEC TR 61968-900:2015-10(en)

IEC TR 61968-900

Edition 1.0 2015-10

TECHNICAL REPORT

colour inside

Application integration at electric utilities – System interfaces for distribution management –
Part 900: Guidance for implementation of IEC 61968-9

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 33.200

ISBN 978-2-8322-2950-7

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

APPLICATION INTEGRATION AT ELECTRIC UTILITIES – SYSTEM INTERFACES FOR DISTRIBUTION MANAGEMENT –

Part 900: Guidance for implementation of IEC 61968-9

FOREWORD

The main task of IEC technical committees is to prepare International Standards. However, a technical committee may propose the publication of a technical report when it has collected data of a different kind from that which is normally published as an International Standard, for example "state of the art".

IEC TR 61968-900, which is a technical report, has been prepared by IEC technical committee 57: Power systems management and associated information exchange.

The text of this technical report is based on the following documents:

Enquiry draft	Report on voting
57/1579/DTR	57/1616/RVC

Full information on the voting for the approval of this technical report can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts in the IEC 61968 series, published under the general title *Application integration at electric utilities – System interfaces for distribution management*, can be found on the IEC website.

The present technical report refers to some ambiguities occurring essentially in IEC 61968-9 and IEC 61968-100 (labelled here as "Warnings"). These issues are being addressed in Working Group 14 of IEC technical committee 57 and will be resolved in the forthcoming new editions of IEC 61968-9 and IEC 61968-100.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

A bilingual version of this publication may be issued at a later date.

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

INTRODUCTION

General

This technical report is intended to help users to interpret IEC 61968-9:2013.

IEC 61968-9 provides a uniform means for performing meter read and control operations over a distributed network.

The most recent version of IEC 61968-9 was published in 2013 and is the second edition. This technical report therefore concentrates on this edition.

Although this technical report concentrates on IEC 61968-9, a portion of this depends on another standard, IEC 61968-100:2013.

The purpose of IEC 61968-9 is to allow heterogeneous components, possibly produced by different vendors, to interoperate with one another. Such components typically include a customer information system (CIS), an outage management system (OMS), a meter data management system (MDMS) and a head-end system (HES).

The messages that are exchanged between the various components are XML documents. IEC 61968-9 defines how these messages are expressed according to the semantics of the operations supported by this standard.

For example, a MDMS might instruct a HES to read the forward bulk energy usage from a set of meters and return the corresponding values back to the MDMS. Alternatively, the HES might be instructed to perform some control operations on a meter – for instance, disconnect the power, reset the readings of that meter and then reestablish the power again. In both cases, IEC 61968-9 defines the precise way in which the appropriate request and response messages are formulated.

While IEC 61968-9 defines the various data items from which the request and response messages are constructed, it is less prescriptive about the corresponding message flows – that is, how a complete message exchange looks. This document provides examples of typical message exchange patterns.

In other words, IEC 61968-9 is informative rather than normative (mandatory) when it comes to describing use cases and message patterns.

IEC 61968-9 does not prescribe the means by which such messages are transmitted from component to component. However, it may be assumed that components communicate with one another either by means of web services (SOAP messages) or over a message bus such as JMS or equivalent.

IEC 61968-9 XML schema definition files

IEC 61968-9 defines many different types of XML message according to the kind of data that are to be transmitted. These message types are referred to as profiles. For example, one such profile corresponds to a meter read request message and another to the corresponding response message.

Annexes H and I of IEC 61968-9:2013 contain listings of various XML schema definition (XSD) files, one for each profile supported by the standard. These constrain the formats of the various allowable XML messages and can be used both to generate sent messages as well as to validate received messages. XSD validation is often a first step in ensuring that received messages are at least syntactically correct, although it does not guarantee that the information in the various fields is always appropriate with regard to the application.

A paper or PDF listing is not a particularly practical way of accessing these XSD documents. However, they are also available in electronic form from the UCAIUG website[1].

Such XSD files may be conveniently examined using a graphical editor such as XML Spy which is a commercial product from Altova GmbH[2]. Open-source tools such as Eclipse[3] offer similar functionality.

Conventions used in this technical report

The examples used in this technical report generally refer to MDMS and HES systems. These names are used for illustrative purposes only. Other system names such as CIS and MDMS or client and server could just as equally well have been chosen.

XML fragments and examples, the names of files and other literal text are shown in a `fixed-width font`.

XML schemas are depicted using screen shots taken from XML Spy. The solid lines represent mandatory elements and the dotted lines represent optional elements. Please see the XML Spy documentation[4] for explanations of the other symbols used.

A sign like this denotes a warning. There are a few areas where special care needs to be taken with IEC 61968-9.

How this technical report is organized

- Clause 3 of this technical report describes the basics of IEC 61968-100 as they relate to IEC 61968-9.
- Clause 4 describes more details concerning IEC 61968-100, especially as to what these have to do with formulating request and response messages and how notifications of errors are communicated. This clause also describes how the standard IEC 61968-9 set of messages may be augmented by implementation-specific messages.
- Clause 5 describes how meters and other objects are named in the IEC 61968-9 world.
- Clauses 6 and 7 respectively describe how meter read operations and meter control operations are carried out.
- Clause 8 shows how a MDMS or HES may be configured with provisioning information.
- Clause 9 discusses some of the less frequently used message exchange patterns, specifically how to schedule actions for execution at some future time and how to cancel them should the need arise.

1 http://iectc57.ucaiug.org/WG14/Part9/Shared%20Documents/Part%209%202Ed/IEC-Part9-Profiles-2nd-Edition%20FDIS.zip

2 http://www.altova.com – XML Spy is the trade name of a product supplied by Altova GmbH. This information is given for the convenience of users of this document and does not constitute an endorsement by the IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

3 http://www.eclipse.org – Eclipse is the trade name of a product supplied by the Eclipse Foundation. This information is given for the convenience of users of this document and does not constitute an endorsement by the IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

4 http://www.altova.com/documents/XMLSpyTutorial.pdf

- Clause 10 provides some details concerning transmitting IEC 61968-9 messages over SOAP (web services) or JMS transports.
- Clause 11 is a detailed reference of the various fields that are used within IEC 61968-9 messages.

APPLICATION INTEGRATION AT ELECTRIC UTILITIES – SYSTEM INTERFACES FOR DISTRIBUTION MANAGEMENT –

Part 900: Guidance for implementation of IEC 61968-9

1 Scope

This part of IEC 61968-9, which is a technical report, is a reference document and, as such, is not always suitable for someone new to the world of meter reading and control. In particular, it assumes significant domain knowledge.

This technical report is a companion document to the official standard. It is written from the viewpoint of a software developer or systems integrator who is tasked with implementing IEC 61968-9. It is not intended as a complete description of this standard. For full details, please refer to IEC 61968-9.

To get the most from this technical report, the user should have a good understanding of XML technologies, in particular of XML schema definitions and of web services.

This technical report contains informative recommendations which may be used to guide implementations of IEC 61968-9 and IEC 61968-100. It does not attempt to be exhaustive. In particular, it focuses on the most common IEC 61968-9 interfaces and assumes the user is using web services or JMS as the underlying transport mechanism. If the user is using other systems or the transport services are something other than web services or JMS, the recommendations in this technical report may be less relevant but perhaps still useful.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 61968-9:2013, *Application integration at electric utilities – System interfaces for distribution management – Part 9: Interfaces for meter reading and control*

IEC 61968-100:2013, *Application integration at electric utilities – System interfaces for distribution management – Part 100: Implementation profiles*

3 IEC 61968-100 basics

3.1 General

The IEC 61968-100 standard is shared across the other IEC 61968 standards, not just IEC 61968-9. In broad terms, whereas IEC 61968-9 is concerned with the *contents* of the various messages, IEC 61968-100 is more concerned with the *construction* of the messages and the *transport* of messages between systems.

An important part of IEC 61968-100 is the `Message.xsd` schema definition file[5]. This describes a message header which forms part of every IEC 61968-9 message. This is

[5] http://iectc57.ucaiug.org/WG14/part100/Shared%20Documents/Common%20Message/Message.xsd

described in further detail in Subclause 3.4. The `Message.xsd` file also describes the entire message envelope – in other words, the overall format of an IEC 61968-9 message including header and other message parts.

IEC 61968-100 also comprises a “generic WSDL” file[6] which further defines how messages are formulated and exchanged when using web services as the transport mechanism. This is described in detail in Subclause 10.2.2.

Before continuing, it should be emphasized that this technical report describes IEC 61968-100 as it specifically relates to IEC 61968-9. In general, however, the principles described here are also applicable to other parts of the IEC 61968 series, especially Parts 3 to 8.

3.2 IEC 61968-100 message exchange patterns

IEC 61968-9 does not completely specify the message exchange patterns – that is, how a set of messages together form a conversation in which one system requests data from another and these are then returned. Such message exchange patterns instead fall under the remit of IEC 61968-100.

A detailed discussion of all the possible message exchange patterns is beyond the scope of this technical report. However, with regard to implementation of IEC 61968-9, there are two broad kinds of message exchange into which all use cases fall.

- **Request and response messages**. This is the classic request-response paradigm. For example, a MDMS system may make an on-demand meter read request to a HES. When the HES has obtained the necessary data, it returns these back to the MDMS along with some indication of success or otherwise of the overall request. Other examples might be for a request to be made to the HES to disconnect the power to a customer or for the meter to be reset to an initial state (if the customer has just moved into a new apartment, say). In these latter examples, the response reflects the success or failure of the request.

  The IEC 61968 series of standards are flexible in defining how a complete conversation may be carried out. For instance, a single request for some meter data may result in several corresponding response messages being generated. This may happen, say, if the meter read requests are for future data. It may also occur if the amount of data being requested is too large to fit into a single response, in which case a series of response messages would be returned that represent the complete meter data delivered in chunks.

- **Unsolicited event messages**. These are made when one entity sends data to another without an initial request having been made. For example, a HES might notify a MDMS when a meter is disconnected. Alternatively, the HES may be configured to read a set of meters at certain times and to report their values without an explicit request message being sent.

  In another example, a typical conversation may begin with a request for an action to take place followed by the corresponding response which in essence says nothing more than “this request appears to be acceptable”. Finally, only when the applications carries out the action at some later time is an unsolicited event message sent to the original requestor to confirm whether the action has succeeded or not.

3.3 IEC 61968-100 message types

3.3.1 General

IEC 61968-100 defines four kinds of message: request messages, response messages, event messages and fault messages. In IEC parlance these message kinds are termed stereotypes.

6 http://iectc57.ucaiug.org/WG14/part100/Shared%20Documents/IEC%2061968-100%202nd%20Edition/Generic%20WSDL%20-%202nd%20Edition/IEC61968-100-Generic.wsdl

- **Request messages** are used for sending queries or commands. For example, a request message might be sent from a MDMS to a HES to instruct the latter to obtain a set of meter readings according to meter identification and other criteria.
- **Response messages** are used for returning the corresponding data or status information. Once it has acquired the required data, for example, the HES sends these back to the MDMS in one or more response messages.

  Response messages are also used to indicate whether a given request succeeded or whether there were any failures in carrying it out.

  Response messages may also be used for sending simple acknowledgements in the context of web services as described in Subclause 10.2.2.
- **Event messages** are used for sending unsolicited data – that is, asynchronous data or status information. For instance, such messages may be sent to notify a MDMS or other entity of an asynchronous event such as a power outage to a meter.

  As noted above, event messages may also be used to send meter readings according to a pre-configured schedule. For example, a HES might maintain its own schedule of meter interrogations and push these data to a MDMS at regular intervals throughout the day.

  An event message that results from a previous request is termed a *consequential event message*.
- **Fault messages** are used for sending notifications of errors that are so severe that the receiving system can perform no meaningful processing. These typically occur within the context of requests made over web services – for instance, as a result of detection of a SOAP fault. More general error notifications – requests that are incomplete or reference non-existent meters and such like – are instead made in a response message. This document does not address fault messages any further.

Any IEC 61968-9 message, whether it is a request message, response message or event message, is composed as an XML document. These different message types are distinguished by the top-level element in the XML document. This must always be one of `<RequestMessage>`, `<ResponseMessage>` or `<EventMessage>` accordingly.

3.3.2 Request messages

Request messages are structured as illustrated in Figure 1. This is an example of a simple message that is sent from a MDMS to a HES to request a meter read.

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>get</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-02T14:16:09Z</Timestamp>
    <MessageID>f4cef283-9a3d-40f3-a3b2-48aa6088ad80</MessageID>
    <CorrelationID>facb121a-b46e-4deb-8188-68a4cbde6746</CorrelationID>
    <Comment>simple meter read request</Comment>
  </Header>
  <Request>
    <GetMeterReadings xmlns="http://iec.ch/TC57/2011/GetMeterReadings#">
      <EndDevice>
        <Names>
          <name>meter1</name>
        </Names>
      </EndDevice>
      <TimeSchedule>
        <scheduleInterval>
          <end>2013-07-25T09:40:00Z</end>
          <start>2013-07-25T09:35:00Z</start>
        </scheduleInterval>
      </TimeSchedule>
    </GetMeterReadings>
  </Request>
</RequestMessage>
```

Figure 1 – Example message for a simple meter read request

At the outermost level there is a `<RequestMessage>` element which, in this example, belongs to the `http://iec.ch/TC57/2011/schema/message` namespace. This namespace corresponds to IEC 61968-100.

Within the `<RequestMessage>` element there is a message header enclosed in a `<Header>` element. All IEC 61968-100 messages – and by implication all IEC 61968-9 messages – must have such a message header which instructs the receiving system how to interpret the remainder of the message.

Of particular importance are the `<Header><Verb>` and `<Header><Noun>` subelements. The combination of verb and noun identify the purpose of the message. Here they have values `get` and `MeterReadings` respectively and thus identify this message as being a meter read request. This is described further in Subclause 3.4. (The other subelements inside the `<Header>` – `<Timestamp>`, `<MessageID>`, `<CorrelationID>` and `<Comment>` – are explained in 3.4.5 and 3.4.6).

The `<RequestMessage>` element also contains a `<Request>` element. This contains the information associated with the request – in this example, a `<GetMeterReadings>` subelement. Other kinds of request – a meter control operation, for example – would contain different information (profiles) within the `<Request>` element.

The `<GetMeterReadings>` element contains the qualifying criteria necessary for making the meter read request. In this case, it specifies a single meter identifier and a time range. In practice, such meter read requests are often much more complicated and contain many more parameters to qualify the precise data being requested – for example, the specific kind of data, a set of time ranges, data that meet certain quality constraints or other criteria.

The `<GetMeterReadings>` element belongs to a different XML namespace – in this example, `http://iec.ch/TC57/2011/GetMeterReadings#`. The schema definition that determines the formulation of the `<GetMeterReadings>` element is defined by IEC 61968-9.

3.3.3 Response messages

A response message is generated as a result of a previous request message. Response messages are structured as illustrated by the example in Figure 2.

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-03T13:08:15Z</Timestamp>
    <MessageID>eed4dac0-c3a5-432a-a9b2-e7c481a9c029</MessageID>
    <CorrelationID>facb121a-b46e-4deb-8188-68a4cbde6746</CorrelationID>
    <Comment>response to a simple meter read request</Comment>
  </Header>
  <Reply>
    <Result>OK</Result>
    <Error>
      <code>0.0</code>
      <level>INFORM</level>
    </Error>
  </Reply>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <MeterReading>
        <Meter>
          <Names>
            <name>meter1</name>
          </Names>
        </Meter>
        <Readings>
          <timeStamp>2013-07-25T09:38:00Z</timeStamp>
          <value>3.1415926</value>
          <ReadingType ref="0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0"/>
        </Readings>
      </MeterReading>
    </MeterReadings>
  </Payload>
</ResponseMessage>
```

Figure 2 – Example response message to a simple meter read request

As shown in this example, a response message contains a <ResponseMessage> element at the top level.

Just like the request message of Figure 1, a response message also contains a <Header> element (3.4). In turn this contains <Header><Verb> and <Header><Noun> subelements.

The <Header><Verb> element is always reply for a response message (3.4.2).

The <Header><Noun> subelement in the response message always matches that of the original request and denotes the kind of message being sent. In this example, the value of MeterReadings denotes that this is a reply to a previous meter readings request.

A response message contains an indication of whether a particular request succeeded or failed and this is included in the <Reply> element. It may be the case that the returned data are spread over several messages in which case this is also indicated. This is described in Subclauses 4.3 to 4.7.

Besides the <Header> and <Reply> elements, the response message also contains a <Payload> element. It is this <Payload> element that contains the actual data being returned as a result of the request – that is, assuming that the request succeeded at least partially. Because the data in the <Payload> element depend on the nature of the reply, the Message.xsd schema definition allows a <Payload> element to hold any kind of data – meter data or whatever is appropriate for the reply.

The <Header><Noun> element specifies the format of the data being sent and this matches the XML schema definition for the <Payload> data. In the example above, the <Header><Noun> is MeterReadings and this corresponds to the namespace of the data within the <Payload> (http://iec.ch/TC57/2011/MeterReadings#). In this example, the data is contained within a <MeterReadings> subelement within the <Payload>. There is a single meter reading containing a time-stamp, value and reading type. The latter describes the precise kind of data measured by the value.

3.3.4 Unsolicited event messages

An unsolicited event message is similar to a response message in that it is used to send data from one system to another. However, unlike a response message, an event message is sent autonomously and not necessarily as the result of some previous request message.

Another difference to response messages is that the noun in the message header need not match that of the original request, if any.

The verb in the message header must be in the past tense and belong to one of the set canceled, closed, changed, created, deleted or executed.

An example of an unsolicited event message is as shown in Figure 3:

```
<EventMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>created</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-13T14:01:32Z</Timestamp>
    <MessageID>df8596fc-816c-42f8-a14f-2a1b3499f33b</MessageID>
    <CorrelationID>ee25e536-2bd4-42c4-947f-0dbf10be1879</CorrelationID>
    <Comment>unsolicited event message containing meter data</Comment>
  </Header>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <MeterReading>
        <Meter>
          <Names>
            <name>meter1</name>
          </Names>
        </Meter>
        <Readings>
          <timeStamp>2013-10-13T14:00:00Z</timeStamp>
          <value>2.71828</value>
          <ReadingType ref="0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0"/>
        </Readings>
      </MeterReading>
    </MeterReadings>
  </Payload>
</EventMessage>
```

Figure 3 – Example unsolicited event message

This message is an example of an unsolicited meter read. Another example might be an unsolicited meter event notification in which HES recognizes that a meter has suffered a power loss and so notifies the MDMS accordingly. The latter case is described more fully in Subclause 6.6.

An event message contains an `<EventMessage>` as the top-level element. It contains `<Header>` and `<Payload>` elements but no `<Reply>` element.

3.4 IEC 61968-100 message content

3.4.1 General

As noted previously, IEC 61968-100 includes the `Message.xsd` XML schema definition file.

This file specifies the overall structure of an IEC 61968 message according to whether the top-level element is a `<RequestMessage>`, `<ReplyMessage>` or `<EventMessage>`. For instance, Figure 4 shows that a `<RequestMessage>` must contain a `<Header>` subelement and may optionally also contain `<Request>` or `<Payload>` subelements. Strictly speaking, although both are technically optional, in practice at least one of the `<Request>` or `<Payload>` subelements are present.

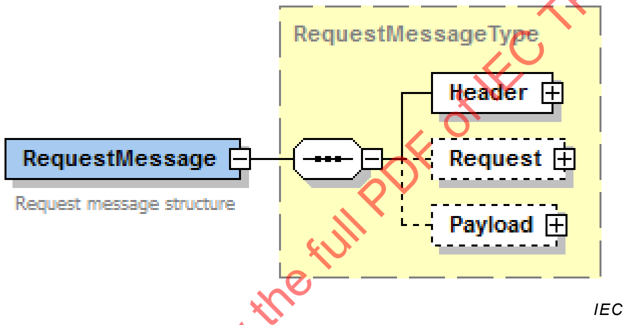


Figure 4 – RequestMessage definition according to Message.xsd

Similarly a response message has the structure depicted in Figure 5. In this case the `<Reply>` element contains status or error information and the `<Payload>` element contains the data being returned as the result of a previous request message.

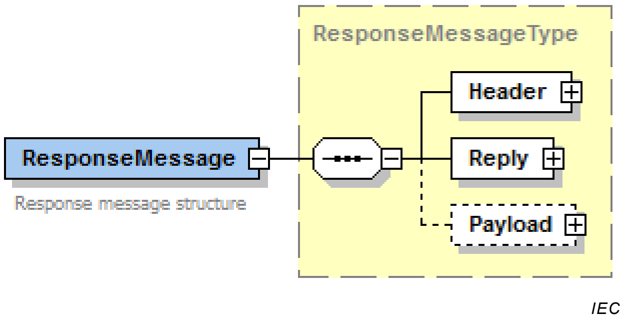


Figure 5 – ResponseMessage definition according to Message.xsd

In the case of an event message, only the payload and header are sent as shown in Figure 6.

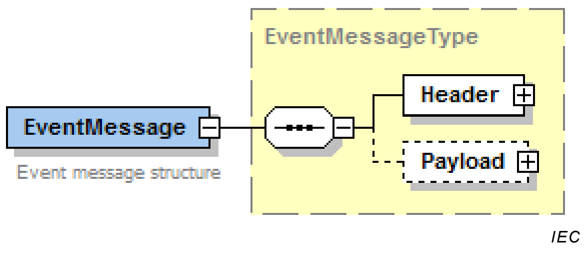


Figure 6 – EventMessage definition according to Message.xsd

3.4.2 Verb and noun elements in the message header

An IEC 61968-9 message must always contain at least a message header with element `<Header>`.

The `<Header>` must contain at least both `<Verb>` and `<Noun>` subelements. As noted above, the definition of this message header is covered by IEC 61968-100 rather than by IEC 61968-9.

The verb contained in the message header must belong to one of the set `cancel`, `change`, `close`, `create`, `delete`, `execute`, `get` for request messages, `reply` for response messages and `canceled`, `closed`, `changed`, `created`, `deleted` or `executed` for event messages. These verbs are always in lower-case.

For meter reading and control operations, the noun belongs to one of the set `MeterReadings`, `EndDeviceControls` or `EndDeviceEvents`. Nouns for operations other than meter reading and control are also possible – for instance, for HES provisioning or configuration. In the case of response messages, the noun is the same as that in the original request.

The combination of verb and noun determines how the message is to be interpreted. For instance, a message with a `<Header><Verb>` of `get` and a `<Header><Noun>` of `MeterReadings` is interpreted by a MDMS or HES as a request for obtaining meter read values (either by addressing the end devices directly or possibly from a database of stored readings).

3.4.3 Get requests

A `get` verb in the message header denotes a query request. The associated information for specifying the query is included in the message `<Request>` element.

For example, the associated data for a meter read request – the meter identifiers, the times of interest for the meter data, the kind of meter data being requested and so on – are all contained in a `<Request><GetMeterReadings>` element as illustrated in Figure 1.

For requests made with a `<Verb>` of `get`, the name of the subelement within the `<Request>` section of the message is the name as that of the noun prepended with `Get`. For example, a `<Verb>` of `get` and a `<Noun>` of `MeterReadings` imply that the name of the subelement is `<GetMeterReadings>`.

IEC 61968-9 and IEC 61968-100 are ambiguous and open to interpretation as to whether the `Get` prefix is required – in this case, whether a `get` request should use a `<GetMeterReadings>` subelement (along with the `GetMeterReadings.xsd` schema definition file) or whether it suffices to use a `<MeterReadings>` subelement instead (together with the `MeterReadings.xsd` schema definition file). However, the conventional understanding is that the former is correct and not the latter. Similar considerations apply to other requests made with a `get` verb.

The Get*Xxx* XSD schema definition files may disappear in future editions of IEC 61968-9 and IEC 61968-100.

3.4.4 Other requests

Verbs other than `get` denote that an action is being requested or that information is being returned as the result of some previous message. For instance, a control message may be sent to cause a switch to be set or a configuration item to be updated. For such requests, the associated information is carried in the `<Payload>` section of the message.

For requests made with a `<Verb>` other than `get`, the name of the subelement within the `<Payload>` section is the same as that of the `<Noun>`. For example, if the `<Verb>` is `reply` and the `<Noun>` is `MeterReadings`, then the name of the subelement within the `<Payload>` section is also `<MeterReadings>`.

3.4.5 The CorrelationID element

Besides the mandatory `<Verb>` and `<Noun>` elements, the IEC 61968-100 header may also contain a `<CorrelationID>` element.

This is useful when a request/response message pattern is being used. When one system sends a request or command to another and sets this element to some value, the expectation is that the responding system will set this element to the same value as that in the original request message. In this way, the requesting system can match received responses to the original requests.

A similar consideration applies for consequential event messages. The `<CorrelationID>` element in such a message corresponds to that in the request that causes the consequential event message to be generated.

The `<CorrelationID>` element should be unique across messages being exchanged between systems. One way of ensuring this is to define this as a 128-bit universally unique identifier (UUID).

3.4.6 Other elements in the message header

Other optional elements that may be set in the message header include the following:

- **`<Revision>`** The profile-specific sections of a message within the `<Request>` and `<Payload>` elements are qualified by namespaces. These namespaces specify the versions of the relevant XSD schema definitions. For those cases where non-standard changes are required, the `<Revision>` element is to be interpreted as a minor version level referring to the payload or request data. The values assigned to this field are non-normative and have meaning only within the local organization.

- **<Timestamp>** This element indicates when the message was produced.
- **<ReplyAddress>** If present, this forces any response messages to be sent to the specified address.
- **<AckRequired>** If this has the value `true`, an explicit acknowledgement message is sent by the recipient system as soon as it receives a request. This element is ignored if web services are being used as the transport mechanism because then an acknowledgement message is sent regardless. This is described further in 10.2.2.
- **<MessageID>** This is a string which can be used to identify the message. This together with the `<Timestamp>` are useful for tracing or logging messages as they are processed through the MDMS, HES or other systems. It has no normative meaning.
- **<Comment>** This element may contain any free text – for example, "`This is example 1`".
- **<Property>** This is a set of name/value pairs that may be used to customize the system behaviour in ways going beyond that defined by the IEC. This is generally used to provide some implementation-specific functionality that goes beyond the standard.

Using the `<Property>` element may impair interoperability. See, for example, the warning in 4.7.3.

3.4.7 The Message.xsd XSD schema definition file

The `Message.xsd` file specifies the subelements permitted inside a `<Header>` element as depicted in Figure 7. Look at the XML fragments in Figure 1 and Figure 2 to see some sample values for these elements.

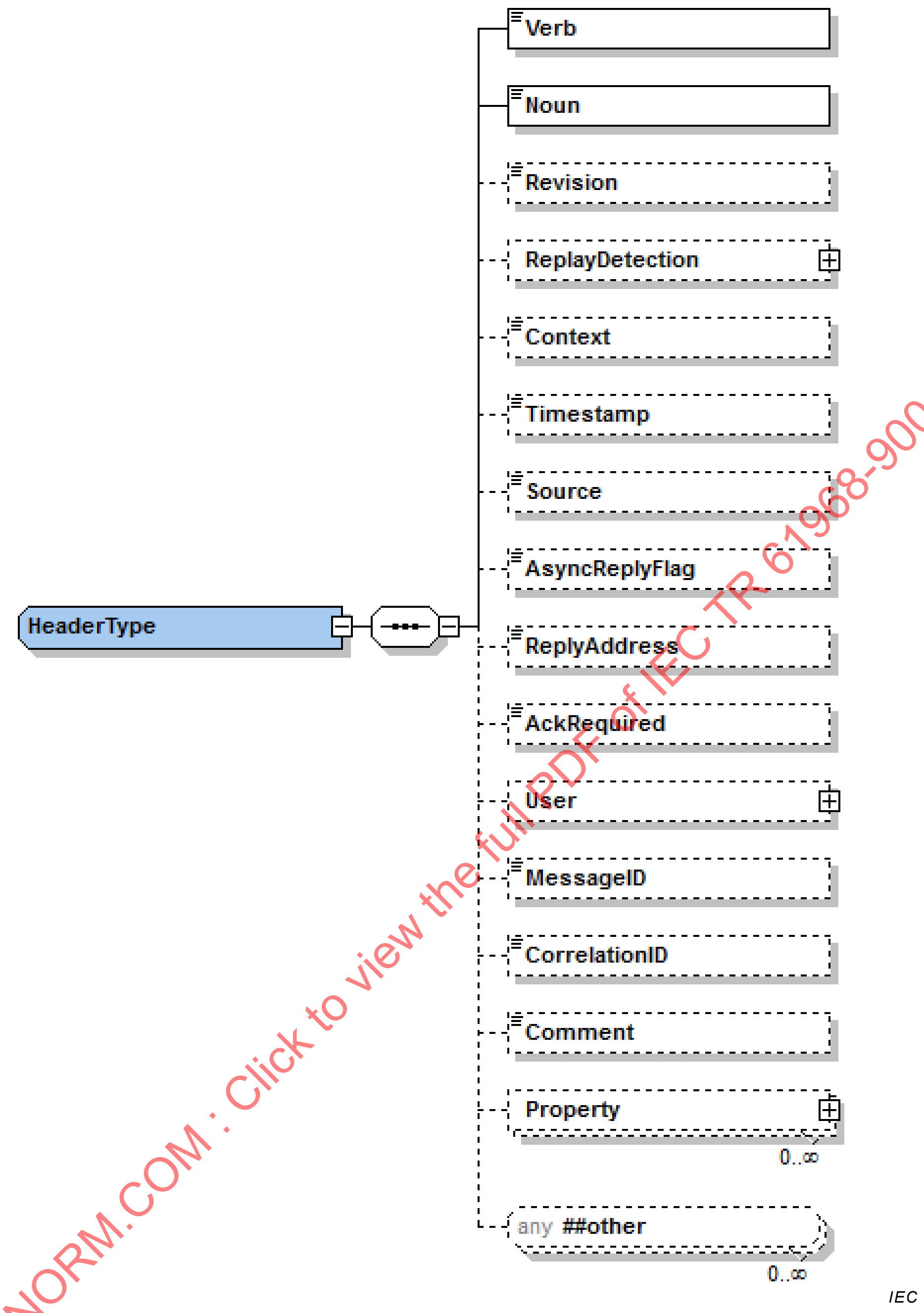


Figure 7 – Subelements allowed inside a <Header> element

4 Request and response messages in detail

4.1 General

There are a few items of note that pertain to IEC 61968-100 request and response messages in general.

4.2 Several potential response messages for each request message

A single request message may cause several response messages to be generated. That is, unlike most messaging paradigms such as a remote procedure call mechanism where there is a strict one to one correlation between requests and responses, IEC 61968-100 allows itself more flexibility.

Although IEC 61968-100 allows for an indication of when the last of several response messages is sent (see 4.7.2), in practice this may not always be possible. Hence it may be difficult for an implementing system to know when it can clear up relevant state information. In general an implementation must rely on generous timeouts before it can safely remove state information.

As noted in 3.4.5, a response message may contain a <CorrelationID> element in the message header, thus allowing for the requesting system to match response messages with the original request.

4.3 Response messages contain a status indication

It is expected that a response message contains an indication of the overall success or otherwise of the request. This is contained in the <Reply> element.

If no errors are encountered in attempting to evaluate the request, this <Reply> element is as shown in Figure 8:

```
<Reply>
  <Result>OK</Result>
  <Error>
    <code>0.0</code>
  </Error>
</Reply>
```

Figure 8 – Example of a success indication in a Reply element

The request may also fail partially or completely in which case the <Reply> element contains the corresponding details. An example of a partial failure is if the request refers to some non-existent meters. An example of a complete failure is if the request cannot be parsed, is missing crucial information or if none of the specified meters exist. This is explained further in 4.5.

If the response data extend over more than one message, the <Reply> element also shows this.

4.4 Response messages containing data and error notifications

According to the Message.xsd schema shown in Figure 5, a response message may contain data, informational conditions and error indications or any arrangement of these.

The data, if present, are placed in the <Payload> element. The informational conditions and error indications, if any, are in the <Reply> element. IEC 61968-100 allows the informational conditions and error indications and response data to be sent in one or over several messages in whatever combination suits the responder best.

In the case of a reply to a meter readings request, therefore, the <Payload> contains the meter read values. The <Reply> element contains error notifications such as the fact that the request may have mentioned a non-existent meter or that the requested data could not be returned for some other reason.

4.5 Specific error indications in the response messages

Error indications are denoted by the inclusion of a <Reply> element where the <Reply><Error><code> subelement has a value other than 0.0, 0.1, 0.2 or 0.3. (These are defined in Annex B of IEC 61968-9:2013 as the response codes for error free or partial success).

The fields of the XML schema corresponding to the <Reply><Error> element are shown in Figure 10.

The <Reply><Error><code> subelement is instead expected to have one of the values defined in Annex B of IEC 61968-9:2013. Table B.10 of IEC 61968-9:2013 contains a list of common values.

The <Reply><Error><level> subelement is optional but, if present, must be one of INFORM, WARNING, FATAL or CATASTROPHIC. A value of FATAL does not necessarily mean that the whole request has failed or even that some irrevocable problem pertaining to the entire request has occurred. It means only that that part of the request – for instance pertaining to a single meter – cannot be answered. For instance, a particular meter mentioned in the request may not exist.

(A particular error code may correspond to an INFORM level in some cases and to a FATAL level in others.)

The <Reply><Error><ID> element indicates to which end point or other object the error notification pertains, possibly also including the name type and name type authority elements of the request. These are placed in idType= and idAuthority= attributes respectively.

The <Reply><Error><ID> element may also include a kind= attribute which is expected to have one of the values name, transaction or uuid. It also includes an objectType= attribute. For the case of a meter that cannot be read, the kind= attribute is name and the objectType= attribute is Meter or UsagePoint according to how the request was formulated (6.3.2). These attributes are shown diagrammatically in Figure 11.

An example of such an error indication is as shown in Figure 9:

```
<Reply>
  <Error>
    <code>2.4</code>
    <level>FATAL</level>
    <reason>no such meter</reason>
    <ID kind="name" objectType="Meter">meter1</ID>
  </Error>
  <Error>
    <code>2.12</code>
    <level>FATAL</level>
    <reason>invalid usagepoint</reason>
    <ID kind="name" objectType="UsagePoint">up2</ID>
  </Error>
</Reply>
```

Figure 9 – Example of error indications in a Reply element (incomplete)

This example is, however, incomplete. The reason for this and the meaning of the <Reply><Result> element are explained in Subclause 4.6.

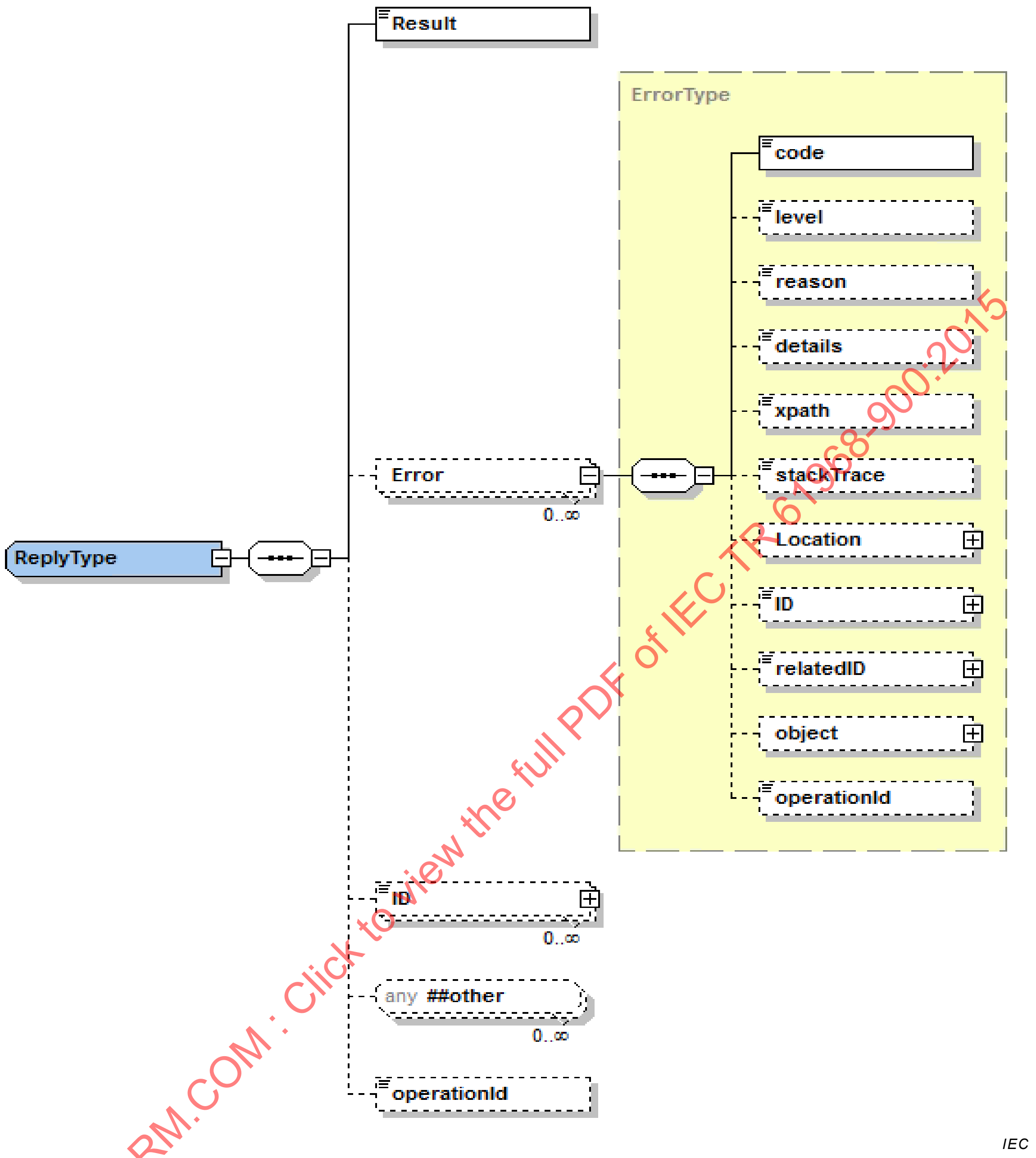


Figure 10 – Subelements allowed inside a <Reply> element

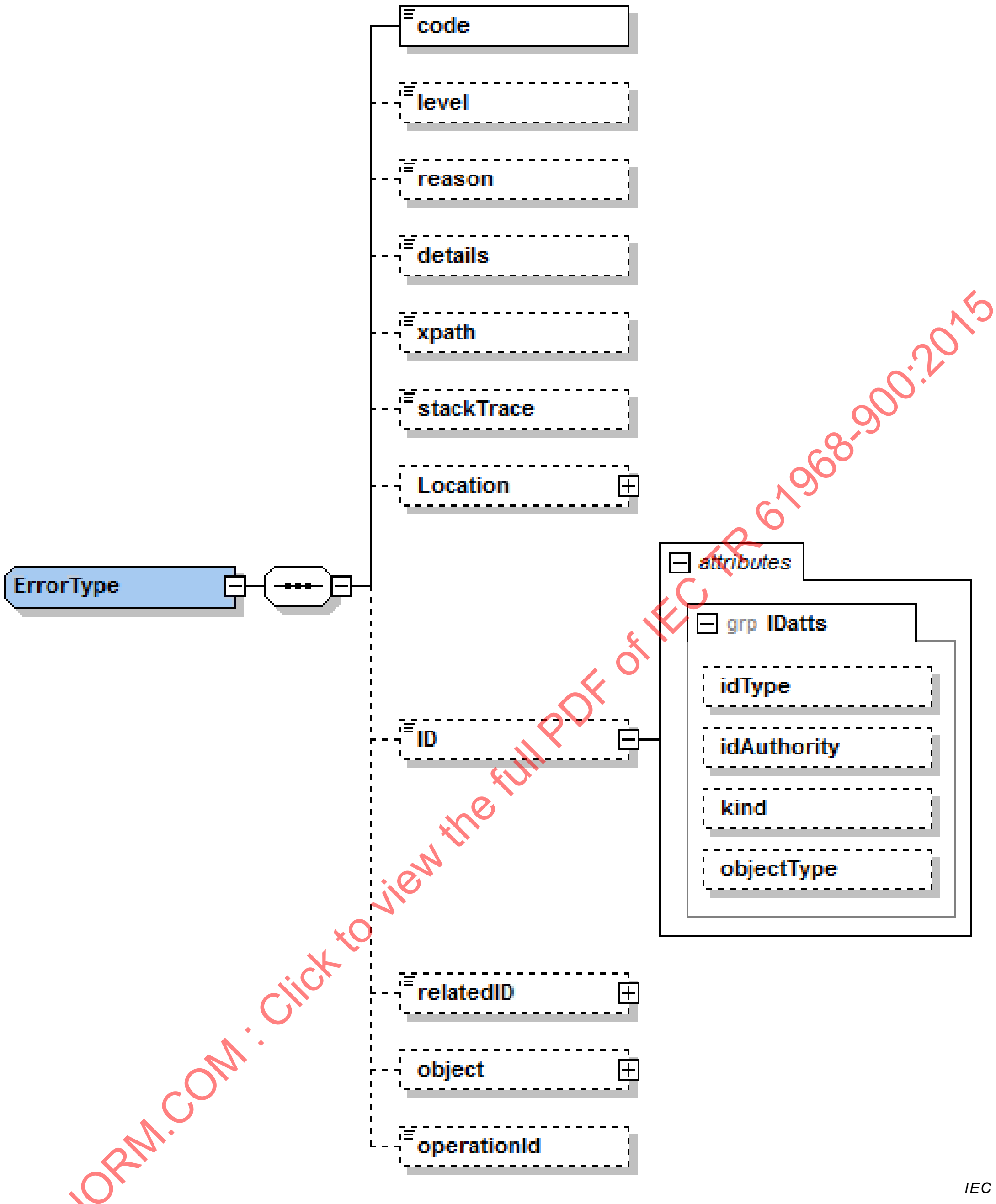


Figure 11 – Subelements allowed inside a <Reply><Error><ID> element

4.6 Implicit indication of success

By convention, `<Reply><Error>` elements are returned in a response message only for those cases where errors are deemed to have occurred. The successful cases are usually not explicitly listed although the standard does not proscribe this.

This implies that, in the case of a multi-message response being sent, a client (receiving) system can determine the successful cases only when all the response messages have been received.

4.7 General error indications in the response messages

4.7.1 General

As previously noted, besides the data being returned along with any possible error notifications, every response message is expected to contain an indication of the overall success or otherwise of the request. The response message may also indicate whether this is the only message being sent or if it is one of several response messages caused by a single request. In the latter case it may also indicate whether it is the last the series of responses or whether further messages are to be expected.

The indication of overall success or failure and the indication of whether further messages are to be expected are placed in the `<Reply><Result>` element, of which there must be exactly one occurrence, together with one or more separate `<Reply><Error>` elements. For example, in Figure 12 the `<Reply>` element signifies the success of the request.

```
<Reply>
  <Result>OK</Result>
  <Error>
    <code>0.0</code>
  </Error>
</Reply>
```

Figure 12 – Example of overall indication of success in a Reply element

Because of the necessity of including these additional `<Reply><Result>` and `<Reply><Error>` elements, the XML fragment shown in Figure 9 should actually read as shown in Figure 13.

```
<Reply>
  <Result>FAILED</Result>
  <Error>
    <code>0.0</code>
  </Error>
  <Error>
    <code>2.4</code>
    <level>FATAL</level>
    <reason>no such meter</reason>
    <ID kind="name" objectType="Meter">meter1</ID>
  </Error>
  <Error>
    <code>2.12</code>
    <level>FATAL</level>
    <reason>invalid usagepoint</reason>
    <ID kind="name" objectType="UsagePoint">up2</ID>
  </Error>
</Reply>
```

Figure 13 – Example of error indications in a Reply element (corrected)

In the above example the `<Reply><Result>` element takes the value `FAILED` rather than `OK`. This is because of the fatal error notifications as explained below.

4.7.2 Setting the <Reply><Result> element

The rules for setting the `<Reply><Result>` element are as follows:

- If the entire response to request message is being provided in a single response message and the response message contains no fatal errors, then set `<Reply><Result>` to `OK`. Also include a single `<Reply><Error><code>` element with value 0.0 as shown in Figure 12. This code is defined by Annex B of IEC 61968-9:2013 as meaning "error free success".

Such a message may also contain informational (non-fatal) conditions. For each informational condition being reported, include a <Reply><Error> element and set the <Reply><Error><code>, <Reply><Error><ID> and other associated <Reply><Error> structure attributes appropriately as described above.

- Otherwise, if the entire response to the request message is being returned in a single message and the response message contains at least one fatal error then set <Reply><Result> to FAILED.

  Such a message may contain a mixture of data items and error notifications. For each fatal error or informational condition being reported, include a <Reply><Error> element and set the <Reply><Error><code>, <Reply><Error><ID> and other associated <Reply><Error> structure attributes as described above.

- Otherwise set the <Reply><Result> element to PARTIAL. This indicates that the responding system is sending multiple response messages to the request message.

  Such messages may contain a mixture of data items and error notifications.

  There must be at least one <Reply><Error><code> element of 0.1 or 0.2. These are defined in Annex B of IEC 61968-9:2013 as the codes for "partial success (additional results conveyed in separate messages)" and "partial success (no further results to follow)" respectively.

  If the responding system cannot determine which is the last message in a set of response messages, then all messages in the set are to be sent with a <Reply><Error><code> of 0.1.

  Also include a <Reply><Error> element and set the <Reply><Error><code>, <Reply><Error><ID> and other associated <Reply><Error> structure attributes (as described above) for each fatal error or informational condition being reported.

4.7.3 Multiple response messages

As noted previously, it is possible for a single request to cause multiple response messages to be generated. As described in 4.7.2, a responding system sets the <Reply><Result> to PARTIAL and the <Reply><Error><code> element to either 0.2 or 0.1, depending on whether this is the last message in the sequence or not.

It might appear that the <Reply><Result> and <Reply><Error><code> elements should suffice for a client (receiving) system to detect when it has obtained all messages belonging to a multiple message response. However, this is true only if the underlying transport mechanism guarantees that messages are received in the same order in which they are transmitted and this may not always be the case. Unfortunately IEC 61968-9 and IEC 61968-100 provide no normative means for a client to detect when it has received all messages belonging to a multiple message response. It is left to the implementer to devise his own scheme. One possible way is to use the <Header><Properties> element to include a sequence number and a "last message" flag.

4.8 Multiple <Request>, <Reply> and <Payload> elements

The Message.xsd schema definition file allows a single message to contain more than one subelement within each <Request>, <Reply> or <Payload> element.

In the case of a request message, multiple subelements within the <Request> element are treated as independent requests. That is, it is treated as though the various requests had been sent in separate messages. Similarly, a reply message or event message containing multiple subelements inside the <Reply> or <Payload> elements is considered as several independent replies or payload data bundled within the same message.

Bundling several response payloads into one message may give rise to ambiguity when trying to match these to the applicable `<Request>` element.

4.9 Implementation-specific messages

An IEC 61968-9 message may be formulated in ways that go beyond the standard. Such a message is valid according to the XML schema definition but its semantics are undefined. This may happen as follows.

- First, as noted above, a message may contain several `<Request>` or `<Payload>` elements within the same message. In such a case, only those elements that match the verb and noun combination in the message header have any meaning according to IEC 61968-9 (Subclause 3.4.2).

  Elements that do not match the given verb and noun combination may also be included in the same message but their semantics are then undefined.
- Second, a `<Noun>` element in the message header may also take on a value not recognized by the IEC. Such a message is legal according to the XSD schema validation but its semantics are not defined. (It is not possible for the `<Verb>` element to be similarly unconstrained because the set of verbs is defined as a XSD enumeration type rather than as a simple string).

In both cases, the meaning ascribed to such messages is implementation specific. Unless there are overwhelmingly good reasons, their usage is not recommended.

Implementation specific payloads should in any case use an organization-specific XML namespace in order to ensure there is no confusion and to emphasize that a non-standard extension is being used.

5 The naming of objects

5.1 General

IEC 61968-9 enforces a consistent scheme for naming objects such as meters and other hardware devices.

5.2 Naming meters

Clearly a meter read request must be able to specify the meters to which the request pertains. The corresponding response to such a request must similarly be able to identify the meters. This implies the necessity of an agreed and unambiguous naming scheme between sending and receiving systems.

For example, as noted in Subclauses 6.3 and 6.5, a meter read request and read response specifies the meters of interest in the `<EndDevice>`, `<EndDeviceGroup>`, `<UsagePoint>` and `<UsagePointGroup>` elements. Each one of these elements has a similar schema definition as depicted in Figure 14.

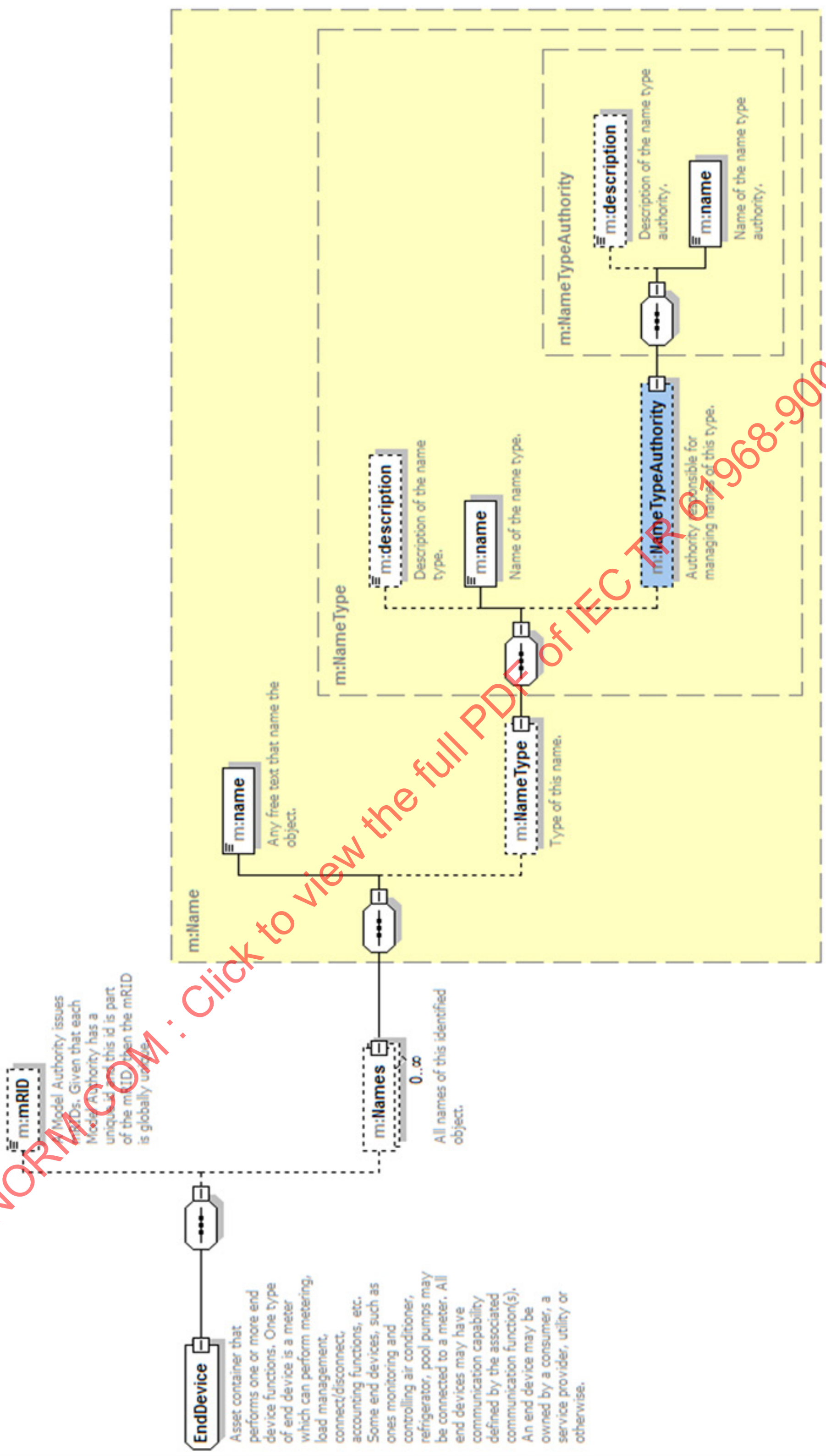

EndDevice
Asset container that performs one or more end device functions. One type of end device is a meter which can perform metering, load management, connect/disconnect, accounting functions, etc. Some end devices, such as ones monitoring and controlling air conditioner, refrigerator, pool pumps may be connected to a meter. All end devices may have communication capability defined by the associated communication function(s). An end device may be owned by a consumer, a service provider, utility or otherwise.
m:mRID
Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part of the mRID, then the mRID is globally unique.
m:Names
0..∞
All names of this identified object.
m:Name
m:name
Any free text that name the object.
m:NameType
Type of this name.
m:NameType
m:description
Description of the name type.
m:name
Name of the name type.
m:NameTypeAuthority
Authority responsible for managing names of this type.
m:NameTypeAuthority
m:description
Description of the name type authority.
m:name
Name of the name type authority.
IEC

Figure 14 – EndDevice definition

According to this schema definition, an EndDevice, for instance, may be specified either as a `<mRID>` (master resource identifier) element or as a `<Names><name>` element. IEC 61968-9 states that a mRID must be provided as a 128-bit universally unique identifier or UUID. In other words, a mRID may not be just a simple meter identifier or serial number. (This generally renders the use of mRIDs as meter identifiers somewhat impractical. There may however be an asset management system or some other system which effectively owns the meters and can assign these mRIDs).

Alternatively, the meter identifier must be specified in an <EndDevice><Names><name> element. If this name is not unique, an optional <EndDevice><Names><NameType><name> element may also be specified to resolve any possible ambiguity. If this still results in a potential conflict, a <EndDevice><Names><NameType><NameTypeAuthority><name> element may also be specified.

In general, a given EndDevice or other object may be known by several names.

Whichever way the names are specified in a request message, the sending and receiving systems must clearly agree on a common and unequivocal scheme for identifying meters.

An example of an EndDevice a meter read request specified using NameType and NameTypeAuthority is shown in Figure 15. In this example the given meter is known by two names.

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>get</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-02T14:16:09Z</Timestamp>
    <MessageID>db511ebc-0047-4174-bace-6e5e2557af88</MessageID>
    <CorrelationID>dc2d7edf-5228-48a2-8fc1-6697eacc5f8f</CorrelationID>
  </Header>
  <Request>
    <GetMeterReadings xmlns="http://iec.ch/TC57/2011/GetMeterReadings#">
      <EndDevice>
        <Names>
           <name>serial-number-2718281828</name>
           <NameType>
             <name>serialId</name>
             <NameTypeAuthority>
               <name>Electric Utility ABC</name>
             </NameTypeAuthority>
           </NameType>
        </Names>
        <Names>
           <name>serial-number-5772156649</name>
           <NameType>
             <name>alternativeSerialId</name>
             <NameTypeAuthority>
               <name>Electric Utility ABC</name>
             </NameTypeAuthority>
           </NameType>
        </Names>
      </EndDevice>
    </GetMeterReadings>
  </Request>
</RequestMessage>
```

Figure 15 – Example of a Meter Read Request with NameType and NameTypeAuthority

5.3 EndDeviceGroups, UsagePoints and UsagePointGroups

Besides specifying meter devices as a <mRID> or as an <EndDevice>, a meter read or control request may also be defined in terms of an EndDeviceGroup, UsagePoint or UsagePointGroup. These share similar structures as shown in Figure 14.

- An <**EndDevice**> corresponds to an actual piece of hardware such as a meter.
- An <**EndDeviceGroup**> is a means for a GetMeterReadings request to refer to a predefined – that is, named – set of end devices. A meter reading request in which an <EndDeviceGroup> is specified refers to all the end devices in that group.
- A <**UsagePoint**> means the location where one or more end devices exist.
- A <**UsagePointGroup**> denotes a set of usage points.

5.4 Naming of other objects

Names may also be used for other kinds of objects within IEC 61968-9. For example, ReadingType values, ReadingQualities, EndDeviceEvents and EndDeviceControls are among the kinds of object that can be associated with a mnemonic name. This is described further in Subclause 6.5.7.

5.5 Provisioning and interrogating a system

Clause 8 describes how a HES or MDMS system may be configured with additional EndDevices or UsagePoints. The corresponding message types (profiles) are EndDeviceConfig and UsagePointConfig respectively. Similarly, a system may be interrogated about the EndDevices and UsagePoints it knows about using GetEndDeviceConfig and GetUsagePointConfig messages.

For EndDeviceGroups and UsagePointGroups, the corresponding message types are EndDeviceGroups, GetEndDeviceGroups, UsagePointGroups and GetUsagePointGroups.

6 Meter read requests and responses

6.1 General

The IEC 61968-9 specification allows for several kinds of meter read requests to a HES or MDMS. These various, overlapping use cases are as follows:

- For a **historical meter read request**, the HES returns whatever data it may have in its database of stored meter values.
- An **interval read** is a set of measurements at regular intervals over some defined period of time. For instance, report the instantaneous forward energy kWh measurements every ten minutes for the period of midday to midnight of the previous day.
- A **load profile read** request is an interval read where the units of measurement are related to energy consumption.
- A **coincident read** request is a request for data not at an explicit time but rather when some other reading was made or when a certain event occurred.
- An **on-demand meter read** instructs the HES to return the requested values available for a given meter by directly asking the meter. This forces the meter data to be obtained from the meter rather than from its internal database of stored meter values.
- An **unsolicited meter read** is made without any explicit request having been made. Instead the metering system initiates the read by itself.

This clause describes how the different message exchange patterns look (6.2). It then describes how the contents of the various request (6.3) and reply messages (6.4) are formulated. On-demand meter read requests are a special case (6.4). Unsolicited meter read messages are also discussed (6.6).

6.2 Message exchange patterns

6.2.1 General

There are three common message exchange patterns relating to how a request for meter data and the corresponding response are made. These are as follows:

- **Request message with a single response message**. This is the simplest case. It is described in 6.2.2.
- **Request message with multiple response messages**. See 6.2.3.
- **Unsolicited meter read**. See 6.6.

The first two of these message exchange patterns are similar to one another. In both cases, the MDMS sends a meter read request to a HES. The HES then validates this request. Assuming that the request message supplies sufficient information to process the request along with valid meter identifiers, the HES then goes to the AMI network to read the corresponding values for the meters in question. As these data are received, the HES sends one or more reply(MeterReadings) response messages back to the MDMS.

(The convention reply (MeterReadings) is used to classify the message according to the verb and noun in the header.)

These two cases may be further subdivided according to whether an on-demand meter read is being made or not. The main difference between an on-demand and other kinds of meter read requests lies in the way the requests are formulated. That is, the message exchange patterns are the same but the details of the request messages are different. This is explained in Subclause 6.4.

6.2.2 Request message with a single response message

The relevant sequence diagram is shown in Figure 16.

The corresponding steps are as follows:

1) The MDMS sends a get(MeterReadings) request to the HES. The details of how this request is put together are explained in Subclause 6.3.
2) The HES validates the request. If the request cannot be processed at all – either because essential information is missing or the request specifies only invalid or non-existent meter identifiers – then it sends a reply(MeterReadings) response message to the MDMS with the `<Reply><Result>` element set to `FAILED` along with the appropriate details (4.7). At this point processing of this request then stops.
3) Otherwise, for those meters that it does know about, the HES sends the appropriate commands over the AMI network to instruct the meters to provide the requested information. Depending on the exact request, this may happen straight away or at some time in the future.
4) The meters carry out these requests and return information back to the HES.
5) The HES consolidates all the meter responses into a single reply(MeterReadings) message which it then sends back to the MDMS. If the latter contains no fatal errors, the `<Reply><Result>` element is set to `OK` (4.3). Otherwise this element is set to `FAILED` (4.5).

The essence of this message exchange pattern is that one request generates a single response message. The latter may contain meter data, error notifications or a combination of both (4.4).

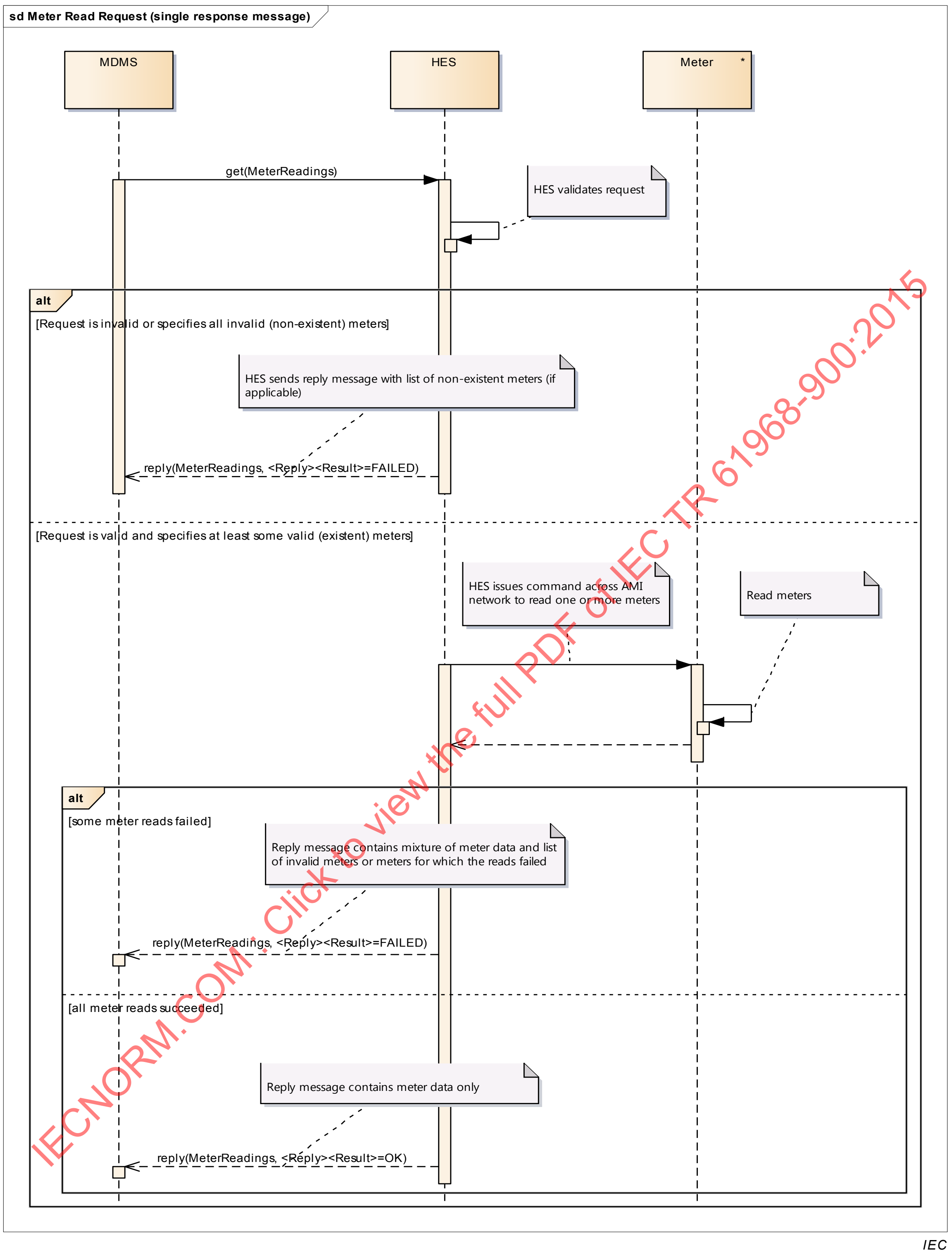

sd Meter Read Request (single response message)
MDMS
HES
Meter
get(MeterReadings)
HES validates request
alt
[Request is invalid or specifies all invalid (non-existent) meters]
HES sends reply message with list of non-existent meters (if applicable)
reply(MeterReadings, <Reply><Result>=FAILED)
[Request is valid and specifies at least some valid (existent) meters]
HES issues command across AMI network to read one or more meters
Read meters
alt
[some meter reads failed]
Reply message contains mixture of meter data and list of invalid meters or meters for which the reads failed
reply(MeterReadings, <Reply><Result>=FAILED)
[all meter reads succeeded]
Reply message contains meter data only
reply(MeterReadings, <Reply><Result>=OK)
IEC

Figure 16 – Message exchange pattern for a meter read request with a single response message

6.2.3 Request message with multiple response messages

The difference between this message exchange pattern and that outlined in 6.2.2 is that, for this case, a request may cause several response messages to be generated.

The relevant sequence diagram is shown in Figure 17.

The steps are as follows:

1) The MDMS sends a get(MeterReadings) request to the HES. The details of this request are explained in Subclause 6.3.
2) The HES validates the request. As in the previous case, it may be that no processing can be performed at all in which case it sends a reply(MeterReadings) response message with the `<Reply><Result>` element set to `FAILED` along with the appropriate details (4.7). At this point processing of this request stops.
3) Otherwise, the HES then sends a reply(MeterReadings) back to the MDMS. This may contain a list of meter-ids for which the HES is unable to provide information, if any (4.7). Because the HES expects to send further reply(MeterReadings) response messages it sets `<Reply><Result>` element set to `PARTIAL` (4.7.3).
4) For those meters that it does know about, the HES sends the appropriate commands over the AMI network to instruct the meters to provide the requested information. Depending on the exact request, this may happen straight away or at some time in the future.
5) The meters carry out these requests and return information back to the HES.
6) The HES sends one or more reply(MeterReadings) response messages back to the MDMS. Again, these have their `<Reply><Result>` elements set to `PARTIAL`. The `<Reply><Error><code>` element has the value `0.1` for all messages except the last for which this element is given a value of `0.2` (4.7.3).

Although this is a slightly more complicated message exchange pattern than that outlined in Subclause 6.2.2, it possibly allows for a simpler implementation inside the HES. The latter can return data back to the MDMS as soon as they are received from the meters. Alternatively, the HES can consolidate data and error notifications into larger messages if it so prefers.

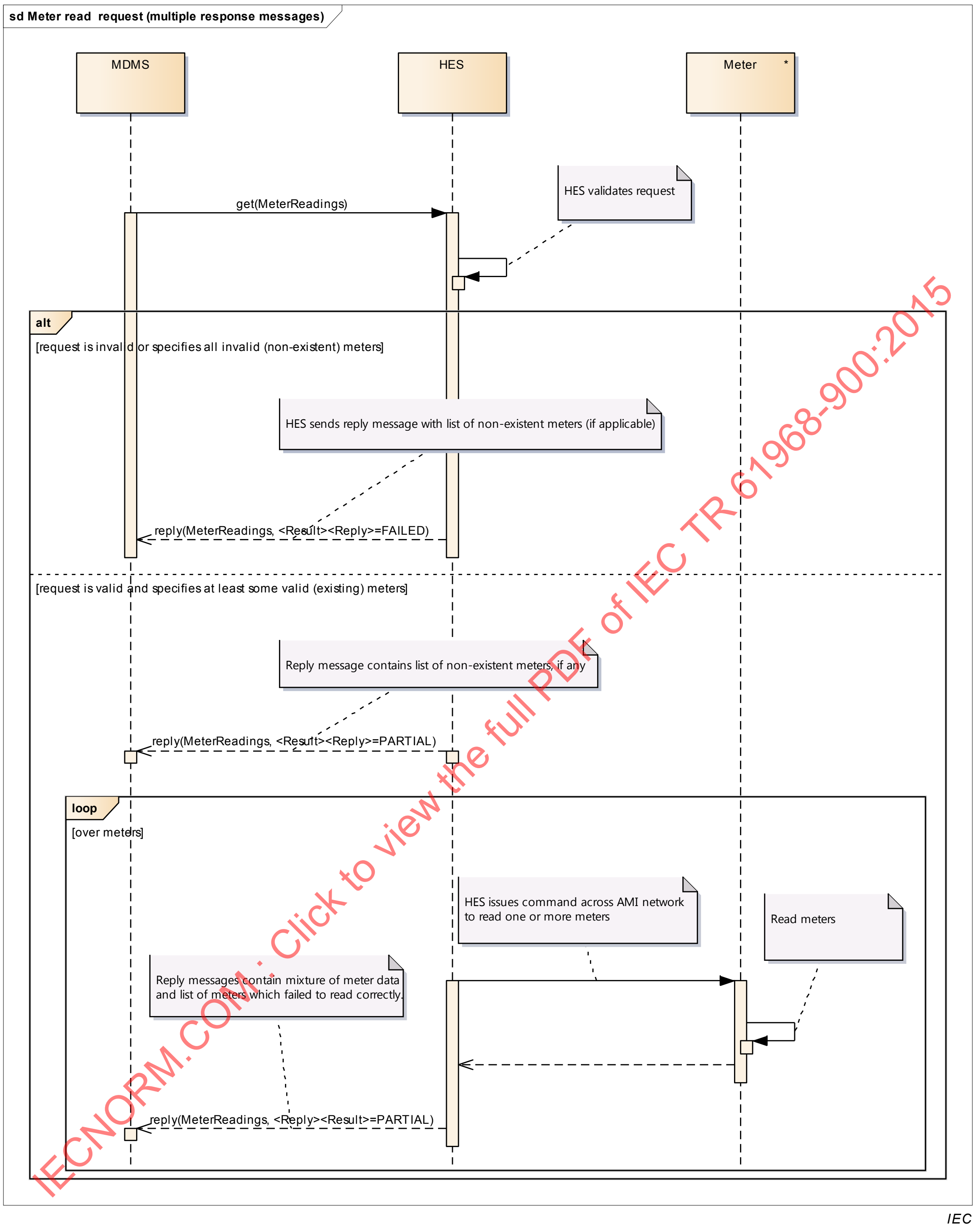

sd Meter read request (multiple response messages)
MDMS
HES
Meter *
HES validates request
get(MeterReadings)
alt
[request is invalid or specifies all invalid (non-existent) meters]
HES sends reply message with list of non-existent meters (if applicable)
reply(MeterReadings, <Result><Reply>=FAILED)
[request is valid and specifies at least some valid (existing) meters]
Reply message contains list of non-existent meters, if any
reply(MeterReadings, <Result><Reply>=PARTIAL)
loop
[over meters]
HES issues command across AMI network to read one or more meters
Read meters
Reply messages contain mixture of meter data and list of meters which failed to read correctly.
reply(MeterReadings, <Reply><Result>=PARTIAL)
IEC

Figure 17 – Message exchange pattern for a meter read request with multiple response messages

6.3 GetMeterReadings request

6.3.1 General

A get(MeterReadings) request message is sent to a HES to instruct the HES to respond with the requested meter read data.

The verb and the noun in the message header are `get` and `MeterReadings` respectively. The data in the `<Request>` element describes what data are to be returned to the requesting system.

The `<Request>` element is expected to contain one or more `<GetMeterReadings>` subelements. In graphical terms each `<GetMeterReadings>` element has a schema definition as shown in Figure 18.

The following subelements specify the meters to which the request pertains:

- **<EndDevice>** This specifies the individual logical identities of the meters for which data are being requested.
- **<EndDeviceGroup>** This specifies the identities of the meters by their group identifiers.
- **<UsagePoint>** This specifies the usage points of the meters.
- **<UsagePointGroup>** This specifies the usage points by their group identifiers.

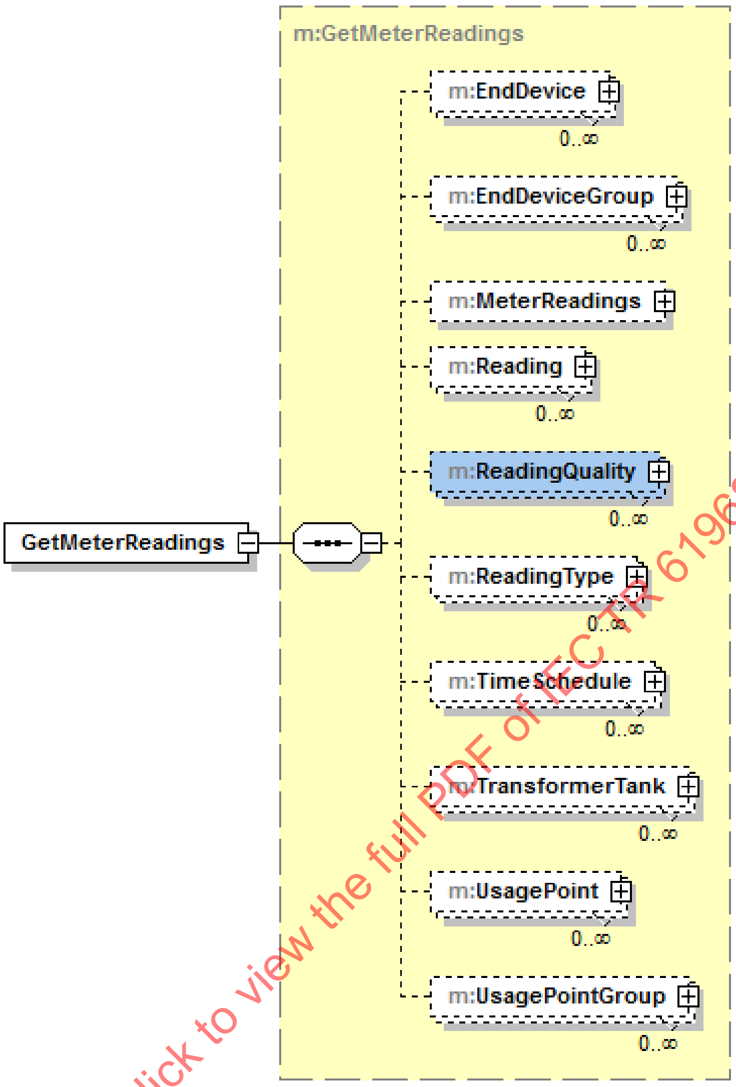
These subelements all share the same kind of naming structure, as explained in Clause 5.

The following subelements specify filtering criteria against which the returned data are matched:

- **<ReadingType>** This specifies the kind of meter data being requested. See Subclause 6.3.3.
- **<ReadingQuality>** This specifies that only meter reading values that match certain quality constraints should be returned (6.3.4).
- **<TimeSchedule>** This specifies the period of interest for which data are being requested (6.3.5).
- **<MeterReadings>** This is used for making a coincident meter read request. As noted above, a coincident meter reading is one whose time is specified when another reading was made or event has taken place.
- **<TransformerTank>** This specifies a meter/transformer relationship, if any. It usually refers to the meter's upstream service transformer secondary winding. This relationship is useful for outage and line-loss (theft) analysis.

The remainder of this clause discusses the various subelements that may appear in a `<GetMeterReadings>` request. These are covered in more detail in 11.2.2.

See the warnings in Subclause 3.4.3 concerning the use of a `<GetMeterReadings>` element rather than `<MeterReadings>` to hold the request data.



IEC

Figure 18 – GetMeterReadings definition according to GetMeterReadings.xsd

6.3.2 Naming of meters

The GetMeterReadings request specifies the meters of interest in the `<EndDevice>`, `<EndDeviceGroup>`, `<UsagePoint>` and `<UsagePointGroup>` elements. Each one of these elements has a similar schema definition as noted in Clause 5.

In almost all cases a GetMeterReadings request will contain at least one such meter identification.

6.3.3 The ReadingType element

The optional `<ReadingType>` element specifies the kind of data that is being asked for in a GetMeterReadings request – for example, forward active energy measured in kWh or 15-minute interval volt-hours for phase A.

The ReadingType has a schema definition as shown in Figure 19. In schema definition terms, a `<ReadingType>` element looks similar to an `<EndDevice>` or other element used for identifying a meter. There is a `<ReadingType><Names><name>` element, optionally qualified by a `<ReadingType><Names><NameType><name>` element which in turn may itself be qualified by a `<ReadingType><Names><NameType><NameTypeAuthority><name>` element. In practice, however, these `<NameType>` and `<NameTypeAuthority>` elements are generally not used.

The `<ReadingType><Name><name>` element contains a string that identifies the kind of data being requested. All ReadingType strings share a similar eighteen-part dotted string in which the individual subparts have their own independent meanings.

For example, the ReadingType string `0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0` is the encoding for bulk forward active energy. (Bulk is a synonym for cumulative. Forward active energy is a measure of the useful rather than reactive electrical energy). The precise meanings attributed to these subparts are described in Annex C of IEC 61968-9:2013. Table C.29 of IEC 61968-9:2013 lists commonly used ReadingType strings.

The third part of such a ReadingType string is the time attribute. In this example, `0` means no time attribute is specified. However, other values such as `1` for ten minutes are possible. The presence of a non-zero time attribute denotes an interval or load profile request.

The first part of a ReadingType string is the time period of interest which is something independent of the time attribute. For instance, a time period of `11` means daily, `13` means monthly and so on. The time attribute and time period of interest may be combined to produce relatively complicated ReadingType specifications. For example, the string `11.8.1.4.1.1.12.0.0.0.0.0.0.0.0.3.72.0` means daily maximum 10 minute delta data forward secondary metered data in kWh.

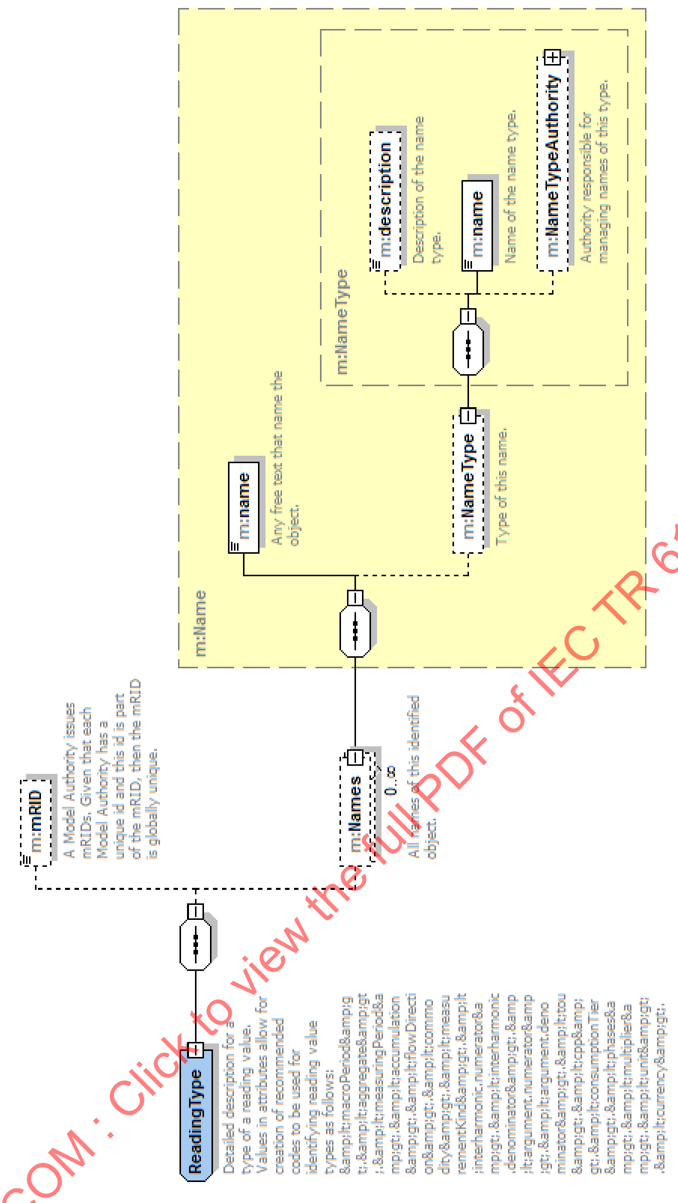

ReadingType
m:mRID
A Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part of the mRID, then the mRID is globally unique.
m:Names
0..∞
All names of this identified object.
m:Name
m:name
Any free text that name the object.
m:NameType
Type of this name.
m:NameType
m:description
Description of the name type.
m:name
Name of the name type.
m:NameTypeAuthority
Authority responsible for managing names of this type.
IEC

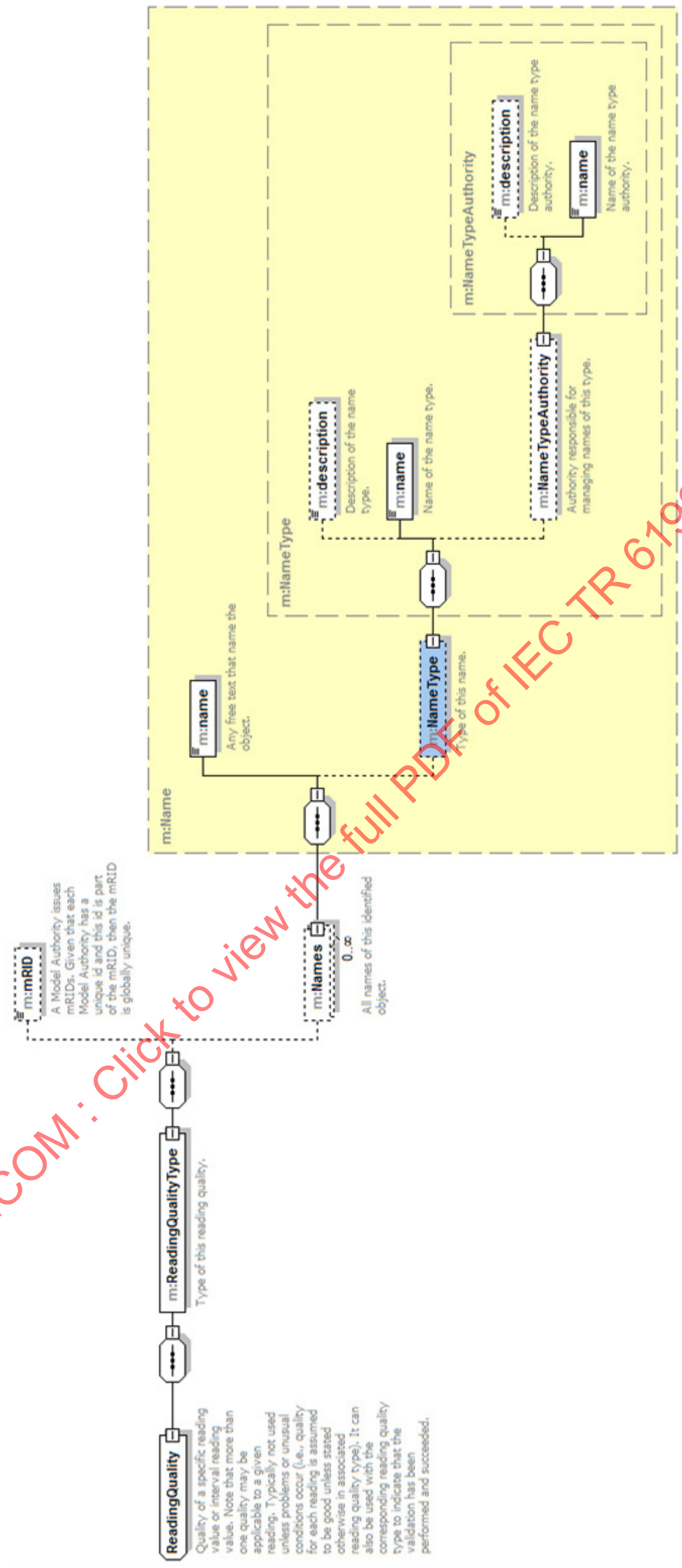
Figure 19 – <GetMeterReading><ReadingType> definition

6.3.4 The ReadingQuality element

The optional <ReadingQuality> element is used to specify that meter readings that match only certain quality constraints are wanted.

The corresponding XML schema definition is shown in Figure 20. Just as for a ReadingType, a ReadingQuality may be specified as a Name, optionally qualified by <NameType> and <NameTypeAuthority> elements. And just as for a ReadingType, these <NameType> and <NameTypeAuthority> elements are unused in practice.

A <ReadingQuality><Name><name> element contains a three part string such as 1.0.0 (data is valid) or 3.7.0 (data was manually edited). Annex D of IEC 61968-9 describes how these ReadingQuality strings are formulated and Table D.15 contains a list of common values.



ReadingQuality
Quality of a specific reading value or interval reading value. Note that more than one quality may be applicable to a given reading. Typically not used unless problems or unusual conditions occur (i.e., quality for each reading is assumed to be good unless stated otherwise in associated reading quality type). It can also be used with the corresponding reading quality type to indicate that the validation has been performed and succeeded.
m:ReadingQualityType
Type of this reading quality.
m:mRID
A Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part of the mRID, then the mRID is globally unique.
m:Names
0..∞
All names of this identified object.
m:Name
m:name
Any free text that name the object.
m:NameType
Type of this name.
m:NameType
m:description
Description of the name type.
m:name
Name of the name type.
m:NameTypeAuthority
Authority responsible for managing names of this type.
m:NameTypeAuthority
m:description
Description of the name type authority.
m:name
Name of the name type authority.
IEC

Figure 20 – <GetMeterReading><ReadingQuality> definition

6.3.5 The TimeSchedule element

The optional `<TimeSchedule>` element specifies the time period of interest for which a meter read request is being made. A request containing such a TimeSchedule can be used, for example, to request historical meter data from a MDMS or HES.

The corresponding XML schema definition is shown in Figure 21. A TimeSchedule contains a `<scheduleInterval>` subelement which in turn contains `<start>` and `<end>` subelements. In common with other time-related elements that may appear in IEC 61968-9 messages, these are expected to be formatted according to the W3C dateTime format[7].

In general a `<TimeSchedule>` element will always be specified in a meter read request. Without such an element, the HES will return all data for the specified meter(s), appropriately constrained according to the `<ReadingType>` and `<ReadingQuality>` elements.

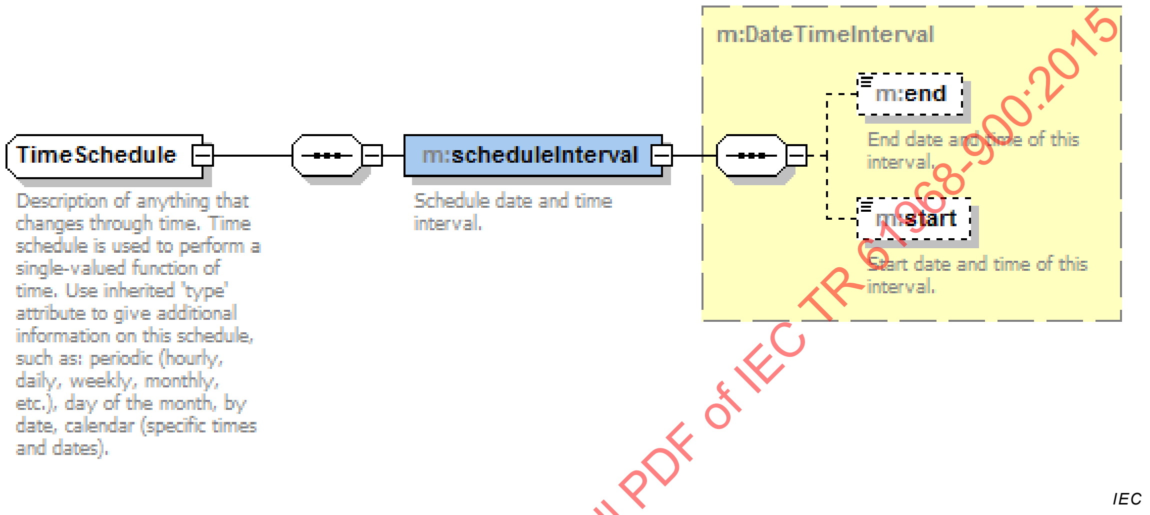

TimeSchedule
Description of anything that changes through time. Time schedule is used to perform a single-valued function of time. Use inherited 'type' attribute to give additional information on this schedule, such as: periodic (hourly, daily, weekly, monthly, etc.), day of the month, by date, calendar (specific times and dates).
m:scheduleInterval
Schedule date and time interval.
m:DateTimeInterval
m:end
End date and time of this interval.
m:start
Start date and time of this interval.
IEC

Figure 21 – <GetMeterReadings><TimeSchedule> definition

IEC 61968-9 provides no normative means for requesting the last known meter read value.

6.3.6 Specifying multiple constraints in a request message

The previous examples show that a request message may ask for meter data that is constrained according to meter identity, reading type, reading quality and time schedule. Other filter constraints are also possible such as specifying coincident meter reads.

The `GetMeterReadings.xsd` schema definition file allows a request message to specify multiple constraints so that a single request may specify several meter identifiers, several reading types and so on. The question then arises, what are the semantics of a message where several such filtering criteria are defined?

The answer is that, when several top-level constraints are specified in a request – and these all refer to different kinds of criteria (meter id, time of interest, reading type, reading quality and so on) – then these constraints are to be interpreted by a logical "AND" operation over the various filters. For example, if a request specifies a certain meter identifier, reading type and time schedule, this means that only those meter readings matching all the specified criteria should be returned. If the request specifies a meter identifier and time schedule but no

7 http://www.w3.org/TR/xmlschema11-2/#dateTime.

reading type, then again all the available data matching this filter should be returned – that is, all data for that meter at the specified time regardless of reading type.

If a request specifies several criteria belonging to the same kind – for instance, several meter identifiers or several reading types, these are interpreted by a logical “OR” operation. For instance, a request specifying several meter identifiers and a single reading type means a request to return the specified data for all those meters and that reading type.

If a request specifies combinations of the various filtering criteria – say, several meter identifiers, several reading types and several reading quality codes – this is interpreted as a request to return data for all combinations of the specified criteria. Thus, for example, if a request specifies meters M1, M2 and M3 and reading types T1 and T2 and reading qualities Q1 and Q2, it may expect to receive data corresponding to (M1, T1, Q1), (M2, T1, Q1), (M3, T1, Q1), (M1, T2, Q1) and all other possible twelve combinations. This is known as the cross-product rule.

There is no means for a single GetMeterReadings element to specify a request for single combinations of the different filtering criteria – for example, only (M1, T1, Q1) and (M3, T2, Q1). If such data are desired, then there are two possibilities to specify such a request. One way is to send two independent request messages. A second way is to combine several GetMeterReadings elements into the same `<Request>` element in a single message (see 4.8). These are then interpreted as if they belonged to separate requests.

The following two examples illustrate this point. Figure 22 shows a request for the forward active energy and forward reactive energy over two meters which corresponds to four data values in the response message (assuming each meter/ReadingType combination yields just one data point).

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message"
  <Header>
    <Verb>get</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-02T14:16:09Z</Timestamp>
    <MessageID>d1d5c311-1e9f-401b-b4b9-7e73d25de404</MessageID>
    <CorrelationID>d8b6c828-e5f6-443e-b872-805ba4e3b2d8</CorrelationID>
  </Header>
  <Request>
    <GetMeterReadings xmlns="http://iec.ch/TC57/2011/GetMeterReadings#">
      <EndDevice>
        <Names>
          <name>meter1</name>
        </Names>
      </EndDevice>
      <EndDevice>
        <Names>
          <name>meter2</name>
        </Names>
      </EndDevice>
      <ReadingType>
        <Names>
          <name>0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0</name>
        </Names>
      </ReadingType>
      <ReadingType>
        <Names>
          <name>0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.73.0</name>
        </Names>
      </ReadingType>
      <TimeSchedule>
       <scheduleInterval>
        <end>2013-07-25T09:40:00Z</end>
        <start>2013-07-25T09:40:00Z</start>
       </scheduleInterval>
      </TimeSchedule>
    </GetMeterReadings>
  </Request>
</RequestMessage>
```

Figure 22 – Example of a request message for two ReadingType codes over two meters

In Figure 23 the request is for data for one ReadingType from meter1 and a different ReadingType from meter2.

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message"
  <Header>
    <Verb>get</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-02T14:16:09Z</Timestamp>
    <MessageID>c6d2cb99-0b0c-4532-ad16-a28c7cabdcbe</MessageID>
    <CorrelationID>cca4968f-9163-4c8e-8fb6-e43a79a74d06</CorrelationID>
  </Header>
  <Request>
    <GetMeterReadings xmlns="http://iec.ch/TC57/2011/GetMeterReadings#">
      <EndDevice>
        <Names>
          <name>meter1</name>
        </Names>
      </EndDevice>
      <ReadingType>
        <Names>
          <name>0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0</name>
        </Names>
      </ReadingType>
      <TimeSchedule>
        <scheduleInterval>
          <end>2013-07-25T09:40:00Z</end>
          <start>2013-07-25T09:40:00Z</start>
        </scheduleInterval>
      </TimeSchedule>
    </GetMeterReadings>
    <GetMeterReadings xmlns="http://iec.ch/TC57/2011/GetMeterReadings#">
      <EndDevice>
        <Names>
          <name>meter2</name>
        </Names>
      </EndDevice>
      <ReadingType>
        <Names>
          <name>0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.73.0</name>
        </Names>
      </ReadingType>
      <TimeSchedule>
        <scheduleInterval>
          <end>2013-07-25T09:40:00Z</end>
          <start>2013-07-25T09:40:00Z</start>
        </scheduleInterval>
      </TimeSchedule>
    </GetMeterReadings>
  </Request>
</RequestMessage>
```

Figure 23 – Example of a request message for two meter/ReadingType combinations

6.3.7 Coincident meter reads

A coincident meter read is one where the read is specified not for a particular time but rather when some other meter read or event happens. For example, a common business process is for a meter reading to be obtained when a meter is installed. Similarly, when a meter is disconnected, it is common practice to capture a final read coincident with the disconnection event.

Coincident meter reads are explained in Subclause 5.3.3 of IEC 61968-9:2013.

6.4 On-demand meter reads

6.4.1 General

As noted above, an on-demand meter read is used for when the HES is being explicitly instructed to read data from the meter. The HES is thereby forced to disregard any data that it may already have in its internal data store.

The message pattern for this use case is similar to the “Request Message with a Multiple Response Messages” described in 6.2.3. The differences lie in the way in which the request is formulated.

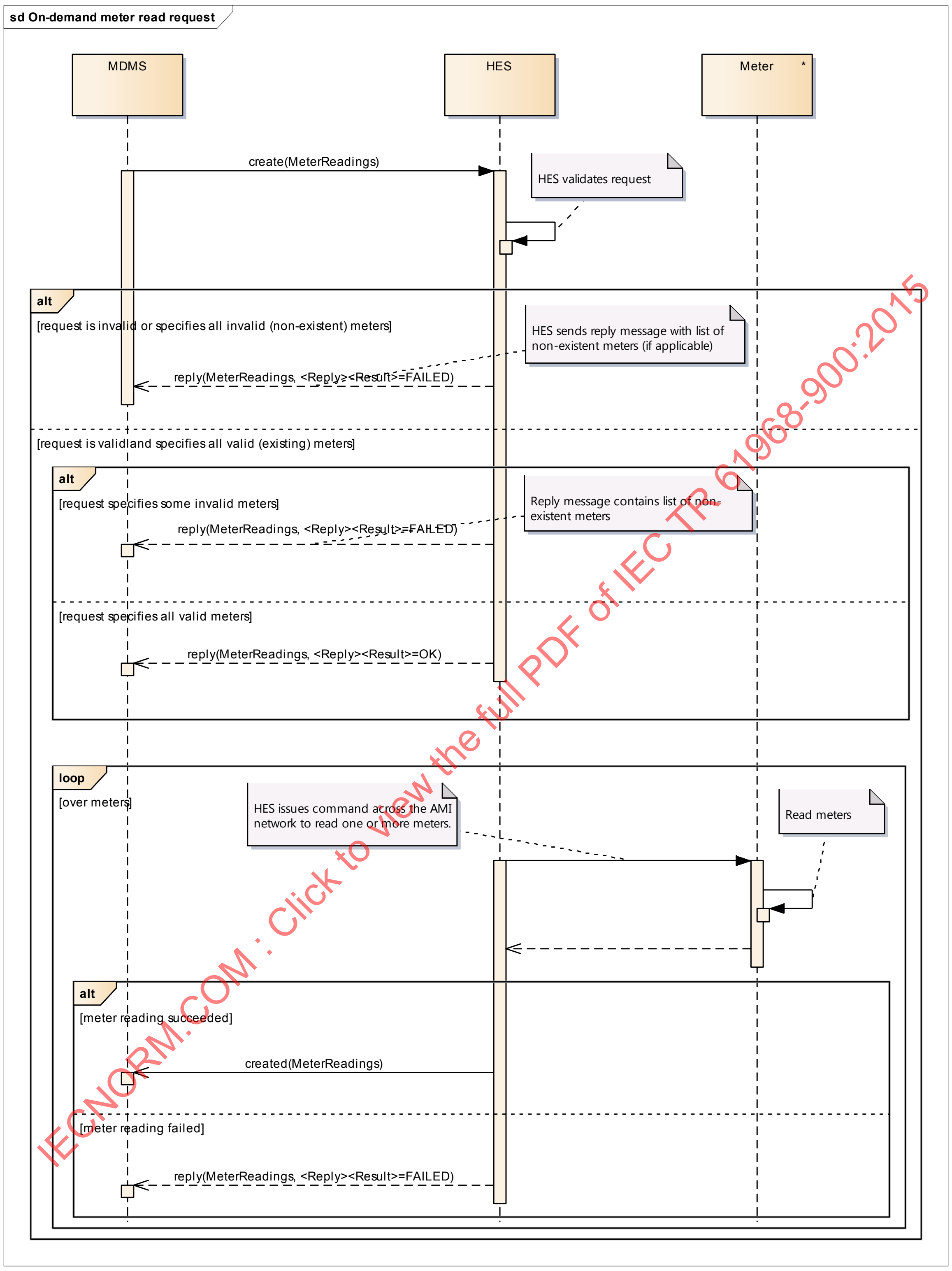
The relevant message exchange pattern for a simple on-demand meter read request message is shown in Figure 24. This is similar to the example in Figure 22 except for the following differences:

- The header verb is `create` instead of `get`.
- The MeterReadings profile is used rather than GetMeterReadings. (A schematic diagram of the MeterReadings profile is shown in Figure 27.)
- The information needed for making the request is carried within the `<Payload>` element rather than a `<Request>` element.
- The meter identfiers are specified within `<Meter><Names><name>` subelements.
- In the common case no `<TimeSchedule>` element is specified. However, if a `<TimeSchedule>` element is provided, this should refer to a future time in which case a meter read is scheduled for that time. See also 9.2.

An example of a create(MeterReadings) message is shown in Figure 25.

The requested data are returned as a sequence of consequential created(MeterReadings) messages rather than as reply(MeterReadings) messages. An example of such a created(MeterReadings) message is shown in Figure 34.

If for whateever reason the meter read fails, a reply(MeterReadings) message is sent back to the requestor with an appropriate error code.


sd On-demand meter read request
MDMS
HES
Meter *
create(MeterReadings)
HES validates request
alt
[request is invalid or specifies all invalid (non-existent) meters]
HES sends reply message with list of non-existent meters (if applicable)
reply(MeterReadings, <Reply><Result>=FAILED)
[request is valid and specifies all valid (existing) meters]
alt
[request specifies some invalid meters]
Reply message contains list of non-existent meters
reply(MeterReadings, <Reply><Result>=FAILED)
[request specifies all valid meters]
reply(MeterReadings, <Reply><Result>=OK)
loop
[over meters]
HES issues command across the AMI network to read one or more meters.
Read meters
alt
[meter reading succeeded]
created(MeterReadings)
[meter reading failed]
reply(MeterReadings, <Reply><Result>=FAILED)

IEC

Figure 24 – Message exchange pattern for an on-demand meter read

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message"
  <Header>
    <Verb>create</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-02T14:16:09Z</Timestamp>
    <MessageID>b21b0c12-9bd5-4f1d-80ca-09ad39e0d34b</MessageID>
    <CorrelationID>b97779c1-c094-406b-8e85-0f8169fa06d2</CorrelationID>
  </Header>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <Meter>
        <Names>
          <name>meter1</name>
        </Names>
      </Meter>
      <Meter>
        <Names>
          <name>meter2</name>
        </Names>
      </Meter>
      <ReadingType>
        <Names>
          <name>0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0</name>
        </Names>
      </ReadingType>
      <ReadingType>
        <Names>
          <name>0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.73.0</name>
        </Names>
      </ReadingType>
    </MeterReadings>
  </Payload>
</RequestMessage>
```

Figure 25 – Example of an on-demand meter read request message

6.4.2 Pinging a meter

An on-demand meter read may also be used to "ping" a meter – that is, to determine whether the meter is supplied with grid-side power. For this purpose, the request message specifies a ReadingType of `0.0.0.0.1.11.0.0.0.0.0.0.0.0.0.109.0` (electricitySecondaryMetered energization status).

The response message contains a `<MeterReading><Readings><value>` element of `1` or `0` according to whether the meter is energized or not.

If the meter cannot be reached or is not responding, an appropriate error code will be returned in the `<Reply><Error><code>` element. See also 4.5.

6.5 MeterReadings response

6.5.1 General

Regardless of how the meter read request is constructed, a response message containing meter read data always follows the same outline.

The verb and the noun in the message header are `reply` and `MeterReadings` respectively.

As noted in Subclause 4.2, a single request message may cause several response messages to be issued. This is particularly relevant for meter read requests. First, the request may cover a wide time range possible including times extending into the future. Clearly such a request can be fully answered only when the time in question has been reached. Second, a request may cause the response to be large and so this must be split up over several response messages. A complete MeterReadings response is shown in Figure 26.

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-03T13:08:15Z</Timestamp>
    <MessageID>9528a597-f87a-42a0-a991-4e7699dd5f1d</MessageID>
    <CorrelationID>9b0f0887-3560-4a19-a4f7-2ea11fd253f6</CorrelationID>
  </Header>
  <Reply>
    <Result>OK</Result>
    <Error>
      <code>0.0</code>
      <level>INFORM</level>
      <reason>single response message</reason>
    </Error>
  </Reply>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <MeterReading>
        <Meter>
          <Names>
            <name>meter1</name>
          </Names>
        </Meter>
        <Readings>
          <timeStamp>2012-07-24T09:38:00Z</timeStamp>
          <value>3.1415926</value>
          <ReadingQualities>
            <ReadingQualityType ref="1.0.0"/>
          </ReadingQualities>
          <ReadingType ref="0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0"/>
        </Readings>
        <Readings>
          <timeStamp>2012-07-24T09:38:00Z</timeStamp>
          <value>0.31415926</value>
          <ReadingQualities>
            <ReadingQualityType ref="1.0.0"/>
          </ReadingQualities>
          <ReadingType ref="0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.73.0"/>
        </Readings>
      </MeterReading>
    </MeterReadings>
  </Payload>
</ResponseMessage>
```

Figure 26 – Example of a response to a meter read request

In this example the complete response message contains several individual readings. Each reading in turn contains a reading type qualification to indicate what kind of data is being returned. Each reading also contains a reading quality. Both the reading types and reading qualities are encoded as `ref=` attributes rather than as subelements.

The overall structure of a MeterReadings response is shown in Figure 27. According to this XSD schema definition, a `<MeterReadings>` element permits `<EndDeviceEventType>`, `<MeterReading>`, `<ReadingQualityType>` and `<ReadingType>` subelements. Of these, the `<MeterReading>` element contains the actual meter data. The others are used to provide mnemonic names for the various quantities as described in 6.5.7.

See Subclause 11.2.3 for a detailed description of the various subelements that may appear within a `<MeterReadings>` element.

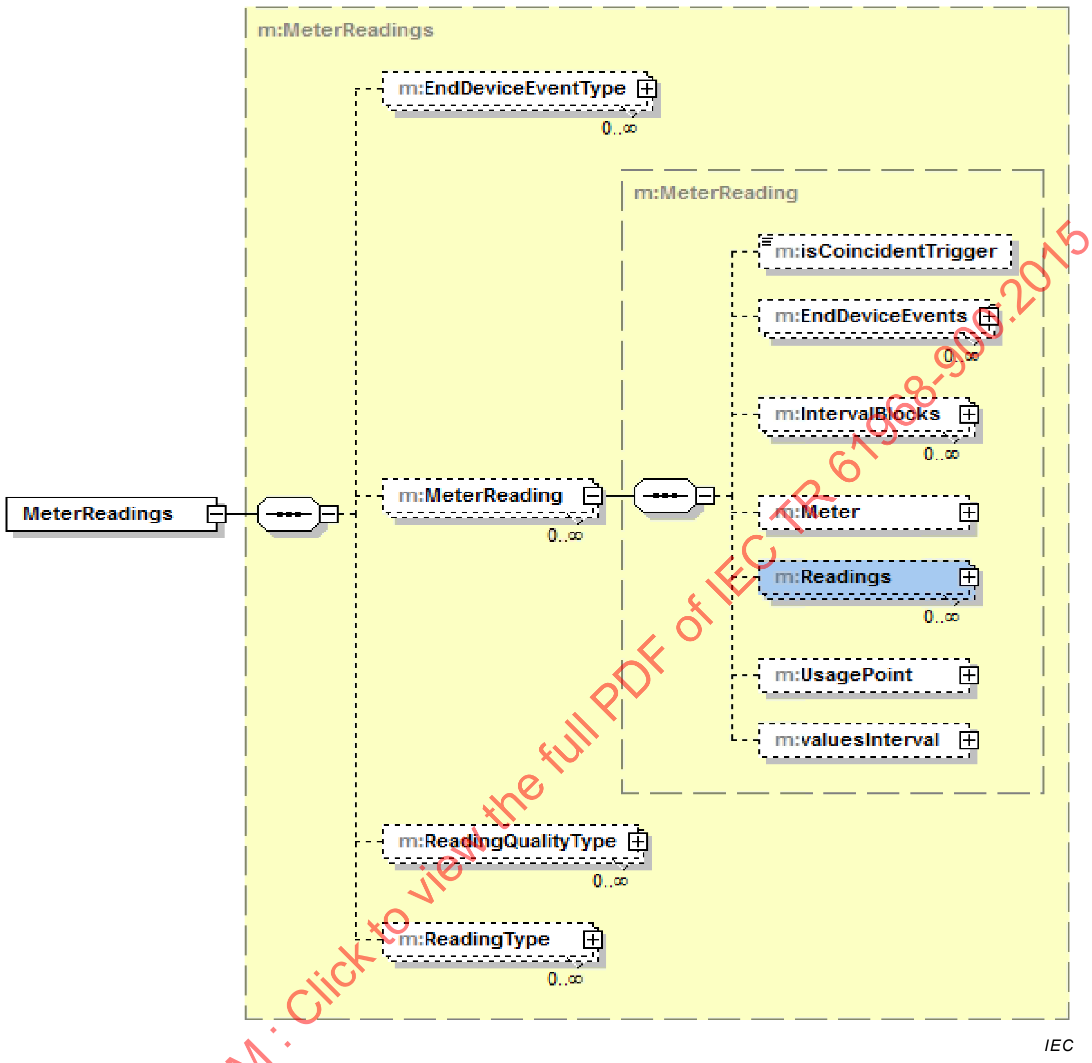


Figure 27 – MeterReadings definition according to MeterReadings.xsd

6.5.2 The <MeterReading><Meter> element

The `<MeterReading>` element contains a `<Meter>` subelement that indicates the particular device for which data are being returned. As shown in Figure 28, a meter contains the same `<Names><name>` element and possibly also `<Names><NameType><name>` and `<Names><NameType><NameTypeAuthority><name>` elements.

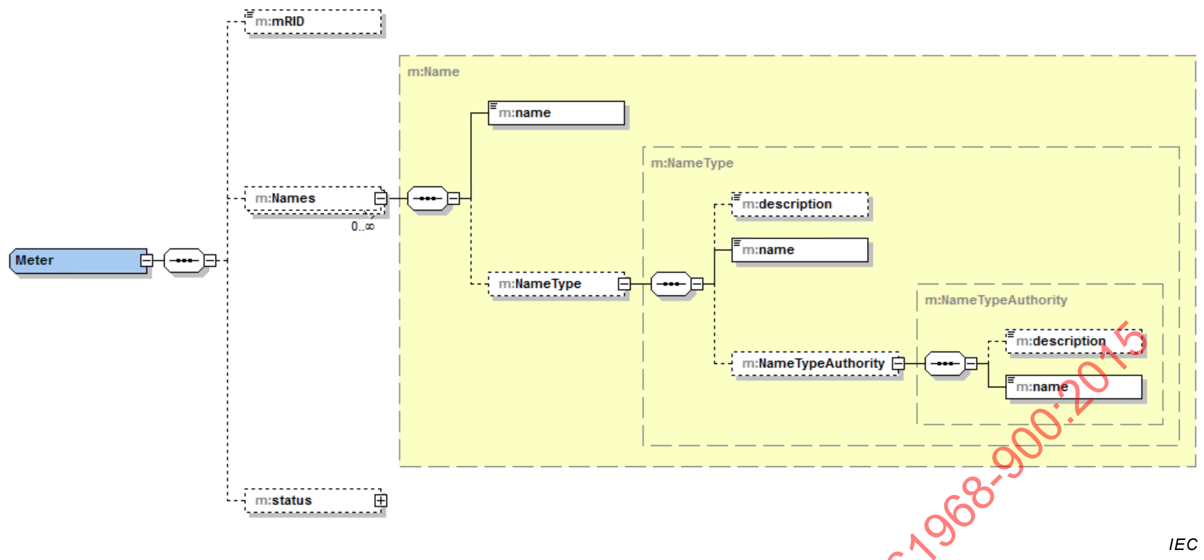


Figure 28 – <MeterReadings><Meter> definition

6.5.3 The <MeterReading><Readings> element

A single meter read response may encode several individual data points. Each data point corresponds to a separate `<Readings>` subelement within the `<MeterReading>` element. The relevant XSD schema definition is shown in Figure 29.

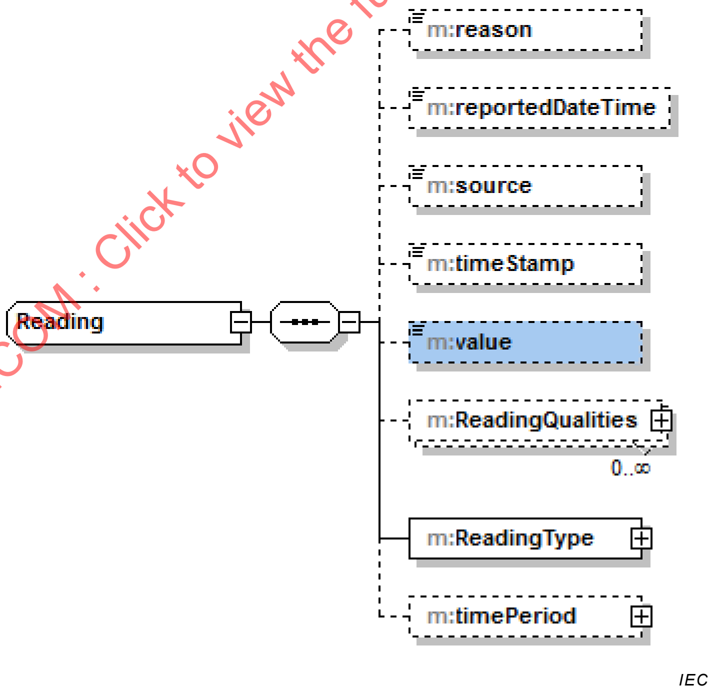


Figure 29 – <MeterReadings><Reading> definition

The `<value>` of the element is the value read from the meter. It is qualified according to the `<timeStamp>` and other elements.

6.5.4 The <MeterReading><Readings><ReadingQualities> element

The XSD schema definition for the ReadingQualitities field is shown in Figure 30. As noted previously and as illustrated in Figure 26, a `ref=` XML attribute is used to encode these values.

See Annex D of IEC 61968-9:2013 for a description of how these 4-part strings are constructed.

A given meter reading may have several reading qualities associated with it. These reading qualities may be assigned a timestamp and together these can define a history of a given reading event.

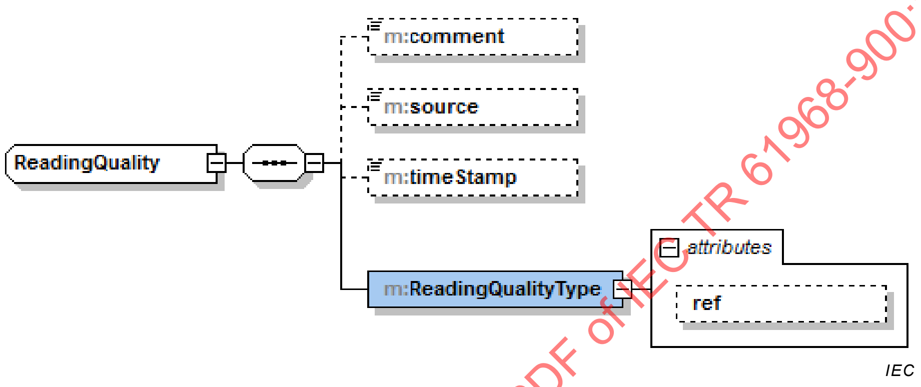


Figure 30 – <MeterReading><ReadingQuality> definition

6.5.5 The <MeterReading><Readings><ReadingType> element

The ReadingType field denotes the kind of data associated with a particular meter reading. A given meter reading may have only one reading type. As noted previously, a `ref=` XML attribute is used to encode these values.

See annex C of the IEC 61968-9 standard for a description of how these 18-part strings are constructed.

6.5.6 The <MeterReading><Readings><IntervalBlocks> elements

The IntervalBlocks field contains a set of meter readings. It provides a more economic means of formulating a reply when all these meter readings share the same ReadingType.

6.5.7 The EndDeviceType, ReadingQualityType and ReadingType elements

The intention of the `<MeterReadings><EndDeviceType>`, `<MeterReadings><ReadingQualityType>` and `<MeterReadings><ReadingType>` elements is to allow alternative names for end device types, reading types and reading qualities to be provided.

For example, as described in 6.5.5, a ReadingType must be specified as an 18-position dotted string. The `<MeterReadings><ReadingType>` allows a more mnemonic name to be defined in terms of the various subelements and which may then be used elsewhere in the response using the `ref=` attribute notation.

Similar possibilities for defining alternative names also apply to the `<MeterReadings><EndDeviceType>`, `<MeterReadings><ReadingQualityType>` elements.

For example, the response message shown in Figure 26 could be more legibly reformulated as shown in Figure 32.

The ability to associate a name with a `<EndDeviceType>`, `<ReadingQualityType>` and `<ReadingType>` element applies only to a MeterReadings response. There is no similar functionality available when making a GetMeterReadings request.

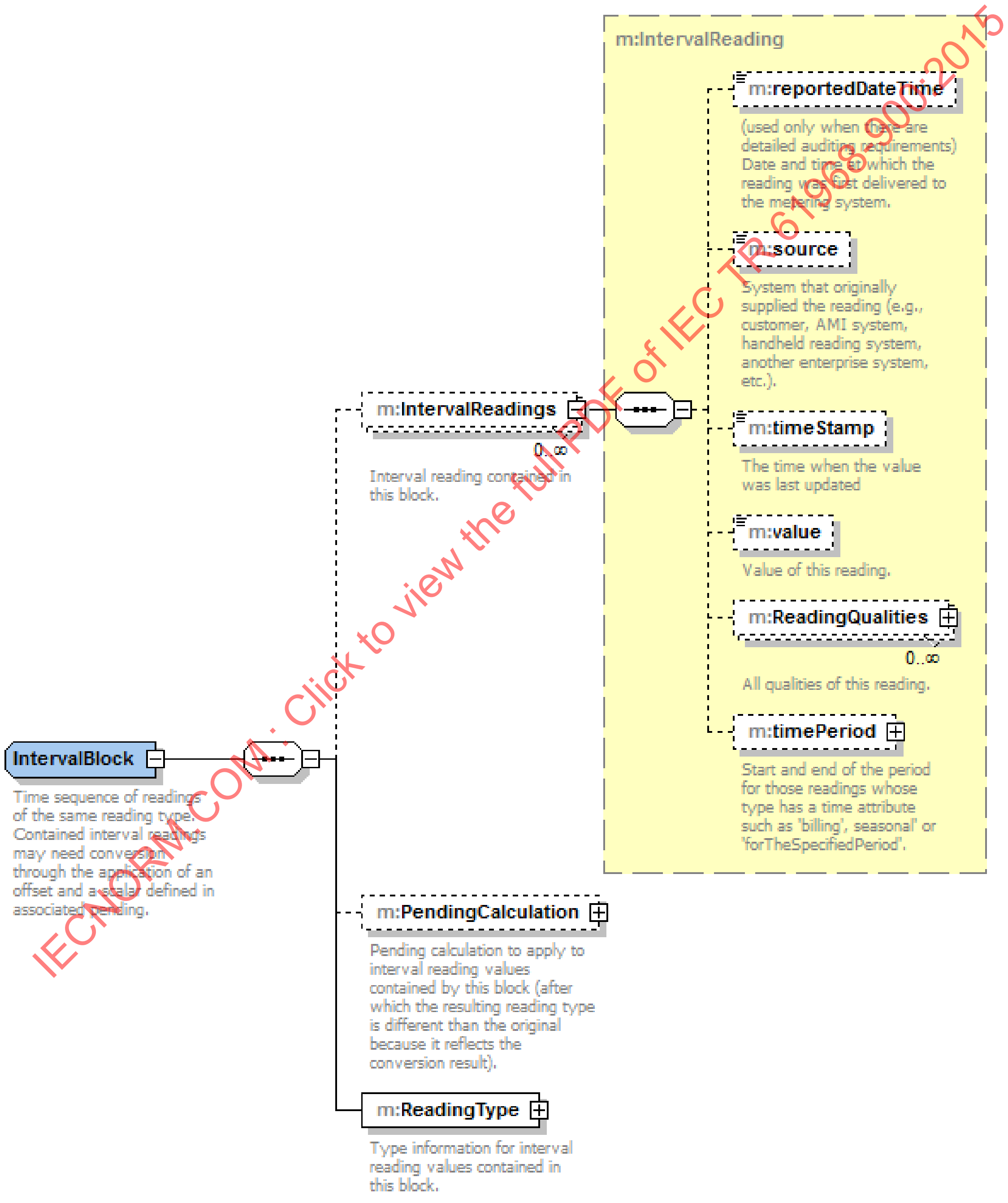

m:IntervalReading
m:reportedDateTime
(used only when there are detailed auditing requirements) Date and time at which the reading was first delivered to the metering system.
m:source
System that originally supplied the reading (e.g., customer, AMI system, handheld reading system, another enterprise system, etc.).
m:IntervalReadings
0..∞
Interval reading contained in this block.
m:timeStamp
The time when the value was last updated
m:value
Value of this reading.
m:ReadingQualities
0..∞
All qualities of this reading.
IntervalBlock
Time sequence of readings of the same reading type. Contained interval readings may need conversion through the application of an offset and a scalar defined in associated pending.
m:timePeriod
Start and end of the period for those readings whose type has a time attribute such as 'billing', seasonal' or 'forTheSpecifiedPeriod'.
m:PendingCalculation
Pending calculation to apply to interval reading values contained by this block (after which the resulting reading type is different than the original because it reflects the conversion result).
m:ReadingType
Type information for interval reading values contained in this block.
IEC

Figure 31 – <MeterReadings><IntervalBlock> definition

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-03T13:08:15Z</Timestamp>
    <MessageID>8c3b4110-91eb-4fa8-9cab-106518be1e54</MessageID>
    <CorrelationID>8dfc122f-6a00-4522-85a3-5d181f7285c8</CorrelationID>
  </Header>
  <Reply>
    <Result>OK</Result>
    <Error>
      <code>0.0</code>
      <level>INFORM</level>
      <reason>single response message</reason>
    </Error>
  </Reply>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <MeterReading>
        <Meter>
          <Names>
            <name>meter1</name>
          </Names>
        </Meter>
        <Readings>
          <timeStamp>2012-07-24T09:38:00Z</timeStamp>
          <value>3.1415926</value>
          <ReadingQualities>
            <ReadingQualityType ref="enddevice, valid data"/>
          </ReadingQualities>
          <ReadingType ref="bulk real forward kWh electricity"/>
        </Readings>
        <Readings>
          <timeStamp>2012-07-24T09:38:00Z</timeStamp>
          <value>0.31415926</value>
          <ReadingQualities>
            <ReadingQualityType ref="enddevice, valid data"/>
          </ReadingQualities>
          <ReadingType ref="bulk reactive forward kWh electricity"/>
        </Readings>
      </MeterReading>
      <ReadingQualityType>
        <category>0</category>
        <subCategory>0</subCategory>
        <systemId>1</systemId>
        <Names>
          <name>enddevice, valid data</name>
        </Names>
      </ReadingQualityType>
      <ReadingType>
        <accumulation>1</accumulation>
        <aggregate>0</aggregate>
        <commodity>1</commodity>
        <consumptionTier>0</consumptionTier>
        <cpp>0</cpp>
        <currency>0</currency>
        <flowDirection>1</flowDirection>
        <macroPeriod>0</macroPeriod>
        <measurementKind>12</measurementKind>
        <measurementPeriod>0</measurementPeriod>
        <multiplier>3</multiplier>
        <phases>0</phases>
        <tou>0</tou>
        <unit>72</unit>
        <argument>
          <denominator>0</denominator>
          <numerator>0</numerator>
        </argument>
        <interharmonic>
          <denominator>0</denominator>
          <numerator>0</numerator>
        </interharmonic>
        <Names>
          <name>bulk real forward kWh electricity</name>
        </Names>
      </ReadingType>
      <ReadingType>
        <accumulation>1</accumulation>
        <aggregate>0</aggregate>
        <commodity>1</commodity>
        <consumptionTier>0</consumptionTier>
        <cpp>0</cpp>
        <currency>0</currency>
        <flowDirection>1</flowDirection>
        <macroPeriod>0</macroPeriod>
        <measurementKind>12</measurementKind>
        <measurementPeriod>0</measurementPeriod>
        <multiplier>3</multiplier>
        <phases>0</phases>
        <tou>0</tou>
        <unit>73</unit>
        <argument>
          <denominator>0</denominator>
          <numerator>0</numerator>
        </argument>
        <interharmonic>
          <denominator>0</denominator>
          <numerator>0</numerator>
        </interharmonic>
        <Names>
          <name>bulk reactive forward kWh electricity</name>
        </Names>
      </ReadingType>
    </MeterReadings>
  </Payload>
</ResponseMessage>
```

Figure 32 – Example of a meter read response with named ReadingType and ReadingQuality elements

6.6 Unsolicited meter reads

6.6.1 General

A HES may also send unsolicited meter reads to a MDMS. Such meter reads are typically configured during the installation of a HES so that at certain defined times it reads the latest meter values from the AMI network and sends these to the MDMS.

6.6.2 Message exchange pattern

The sequence diagram for this message exchange pattern in shown in Figure 33.

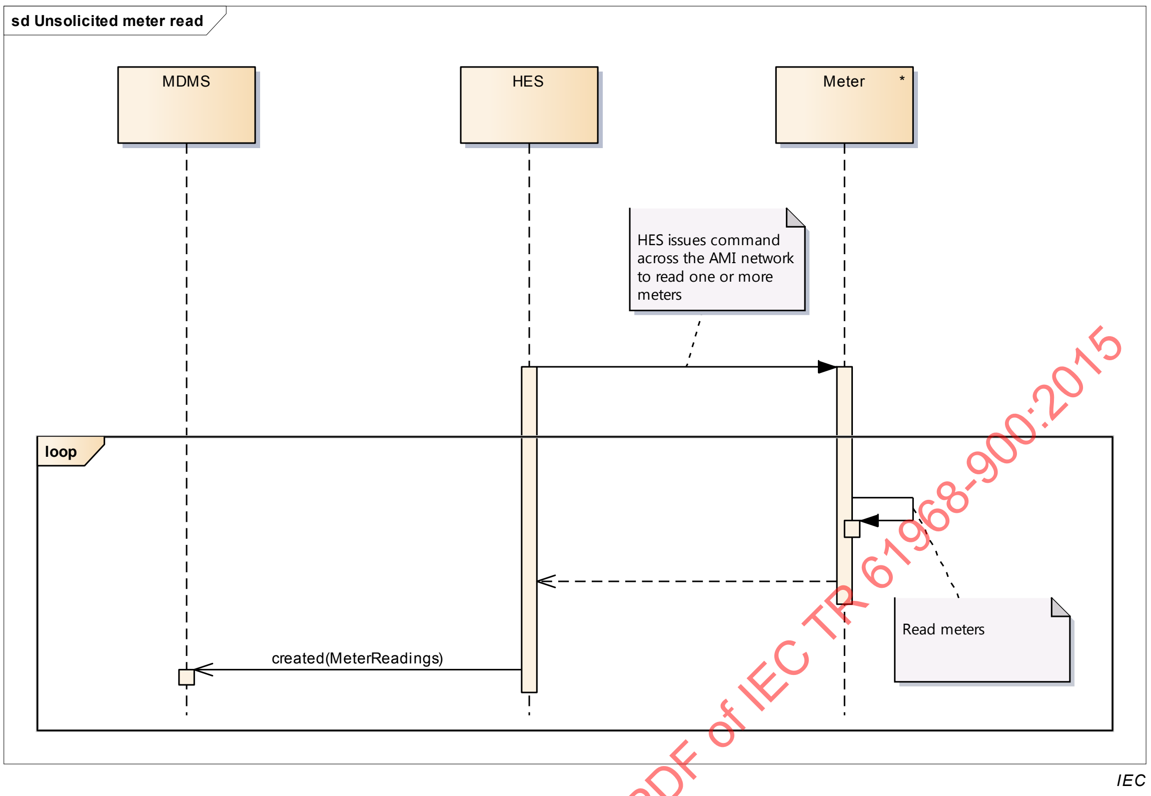

sd Unsolicited meter read
MDMS
HES
Meter *
HES issues command across the AMI network to read one or more meters
loop
Read meters
created(MeterReadings)
IEC

Figure 33 – Message exchange pattern for a set of unsolicted meter reads

An unsolicited meter reading message contains a header with a verb of `created` and a noun of `MeterReadings`. The meter readings are sent inside the `<Payload>` section of the message as shown in Figure 34.

```
<EventMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>created</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-13T14:01:32Z</Timestamp>
    <MessageID>d1d5c311-1e9f-401b-b4b9-7e73d25de404</MessageID>
  </Header>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <MeterReading>
        <Meter>
          <Names>
            <name>meter1</name>
          </Names>
        </Meter>
        <Readings>
          <timeStamp>2012-10-13T14:00:00Z</timeStamp>
          <value>2.71828</value>
          <ReadingType ref="0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0"/>
        </Readings>
      </MeterReading>
    </MeterReadings>
  </Payload>
</EventMessage>
```

Figure 34 – Example of an unsolicited meter read message

The `<Payload>` element contains one or more `<MeterReadings>` subelements. In turn these contain further subelements whose meanings have been described in 6.5.

The XML schema validation does not enforce the inclusion of a time zone in each timestamp. However, in order to avoid potential misinterpretation, it is strongly recommended that each timestamp include a time zone.

6.6.3 Missing reads

When a MDMS or HES is configured to publish a given set of data on a periodic basis, but is unable (at the time) to deliver on all its commitments, it may publish a message in which the value is missing but other data is present to describe the reading. In the example below, there are two readings being published. Both are assigned to the same dateTime reading timestamp but one is missing.

```
<EventMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>created</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-13T14:01:32Z</Timestamp>
    <MessageID>84b59555-8e43-4dc5-80e4-7640a8ec69ab</MessageID>
  </Header>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <MeterReading>
        <Meter>
          <Names>
            <name>meter1</name>
          </Names>
        </Meter>
        <Readings>
          <timeStamp>2012-10-13T14:00:00Z</timeStamp>
          <value>2.71828</value>
          <ReadingType ref="0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0"/>
          <!--bulkQuantity forward electricitySecondaryMetered energy (kWh)-->
        </Readings>
        <Readings>
          <timeStamp>2012-10-13T14:00:00Z</timeStamp>
          <!--The <value> element is absent -->
          <ReadingType ref="0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.73.0"/>
          <!--bulkQuantity forward electricitySecondaryMetered energy (kVArh)-->
        </Readings>
      </MeterReading>
    </MeterReadings>
  </Payload>
</EventMessage>
```

Figure 35 – Example of a missing reading in a published message

The example in Figure 35 shows the <value> element being omitted. Other systems may choose to publish an empty element, <value/>.

The publisher is not saying much about the missing read at this point. It may be possible to recover the missing read if the publisher is asked for it specifically as a historical reading.

The publisher may be more specific about the missing reading. Perhaps the publisher knows that this reading is lost forever. It can say so with a ReadingQuality in the message. The example below shows a series of readings being published. They all have a common timestamp but the publisher knows that one of the readings is a known missing (and unrecoverable) reading.

```
<EventMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>created</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2012-10-13T14:01:32Z</Timestamp>
    <MessageID>831e841d-2890-4f17-bf63-0a16c4f35178</MessageID>
  </Header>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <MeterReading>
        <Meter>
          <Names>
            <name>meter1</name>
          </Names>
        </Meter>
        <Readings>
          <value>123.45678</value>
          <ReadingQualities>
            <ReadingQualityType ref="2.2.32"/>
            <!--Outage during interval-->
          </ReadingQualities>
          <ReadingQualities>
            <ReadingQualityType ref="2.4.2"/>
            <!--Partial interval-->
          </ReadingQualities>
          <ReadingType ref="0.0.7.4.1.1.12.0.0.0.0.0.0.0.0.3.72.0"/>
          <!--sixtyMinute deltaData forward electricitySecondaryMetered energy (kWh)-->
        </Readings>
        <Readings>
          <ReadingQualities>
            <ReadingQualityType ref="2.2.32"/>
            <!-- Outage during interval-->
          </ReadingQualities>
          <ReadingQualities>
            <ReadingQualityType ref="2.5.259"/>
            <!-- KnownMissingRead-->
          </ReadingQualities>
          <ReadingType ref="0.0.0.6.0.1.54.0.0.0.0.0.0.0.0.0.29.0"/>
          <!-- indicating electricitySecondaryMetered voltage-rms (V)-->
        </Readings>
        <valuesInterval>
          <end>2012-10-13T14:00:00Z</end>
        </valuesInterval>
      </MeterReading>
    </MeterReadings>
  </Payload>
</EventMessage>
```

Figure 36 – Example of a known missing reading in a published message

The example in Figure 36 describes a scenario where, at the top of the hour a series of readings were to be taken by the meter and published. Unfortunately an outage occurred during the process. The outage had the effect of shortening the measurement of an interval, but a scheduled read at the end of the interval was lost and can no longer be recovered.

6.6.4 Unsolicited MeterReads together with EndDeviceEvents

The `<MeterReading>` element may incorporate optional `<EndDeviceEvents>` and `<Readings>` subelements. These allow alarms to be published asynchronously as well as meter readings. There is also an `EndDeviceEvent.xsd` profile file which can be used to publish stand-alone events as a separate and unrelated activity. Publication of stand-alone asynchronous events is described in 7.5.

Figure 37 shows an example of a message with both a meter reading and an event.

```
<EventMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>created</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2013-06-26T17:35:00Z</Timestamp>
    <MessageID>23bbf080-bd7d-4897-a93f-815c8b56bc6d</MessageID>
  </Header>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <MeterReading>
        <EndDeviceEvents>
          <createdDateTime>2013-06-26T17:30:46Z</createdDateTime>
          <EndDeviceEventType ref="3.26.131.223"/>
          <!-- electricMeter.power.phaseAVoltage.sagStarted -->
        </EndDeviceEvents>
        <Meter>
          <Names>
            <name>meter1</name>
          </Names>
        </Meter>
        <Readings>
          <timeStamp>2012-06-26T17:30:47Z</timeStamp>
          <value>100.2</value>
          <ReadingType ref="0.0.0.6.0.1.54.0.0.0.0.0.0.0.128.0.29.0"/>
          <!-- indicating electricitySecondaryMetered voltage-rms phaseA (V)-->
        </Readings>
      </MeterReading>
    </MeterReadings>
  </Payload>
</EventMessage>
```

Figure 37 – Example of a message showing both an event and a reading

6.7 More about timestamps and interval data

6.7.1 General

There are several opportunities within the schema to present timestamp data.

The `<timeStamp>` near the `<value>` should be used when each timestamp is different. However, if a series of Readings all share the same timestamp, this timestamp may be placed in the `<MeterReading><valuesInterval><end>` element.

Figure 38 shows the `<valuesInterval><start>` and `<valuesInterval><end>` elements within the `<MeterReading>` element.

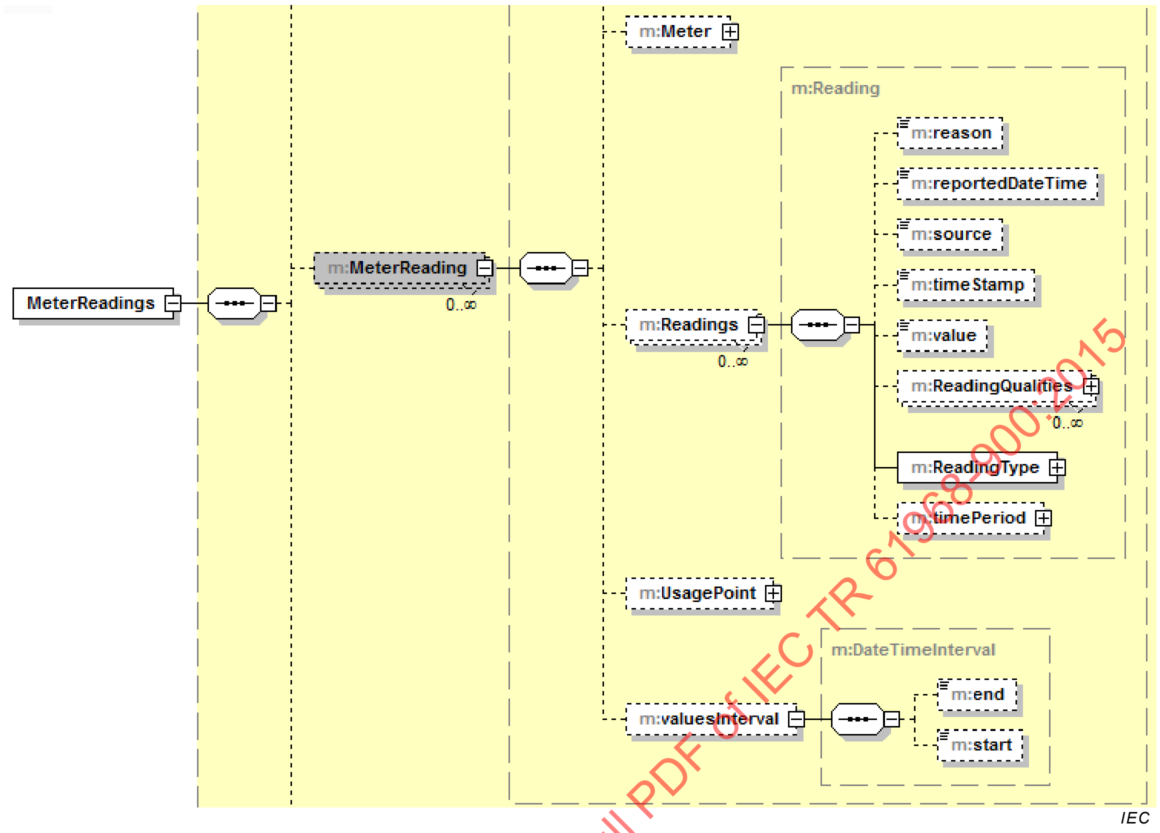


Figure 38 – Reading Timestamps

6.7.2 Interval data

There are two types of interval data. One type measures “delta data” over a timespan. The other type takes a snapshot of an “indicating” value at a specific point in time. Delta data is typically measured in the meter so that at the end of the interval (for instance, the top of the hour) a dial reading is taken and the difference between the last sample (taken at the end of the last interval) is calculated and stored as the delta. This implies that the specified timestamp would be exclusive of the start of the interval, and inclusive of the end of the interval. For example a request or reply specifying a timespan of “`<start>2013-10-13T14:00:00Z</start>`” and “`<end>2013-10-13T15:00:00Z<end>`” describes energy collected starting just after the instant of 14:00 and ending at 15:00 including the instant.

(An interval like this could be written in ISO_31-11 bracket notation [8] in which parentheses are used to indicate a timestamp exclusive of the instant, and square brackets inclusive of the instant. In this example, the interval would look like `(14:00 15:00]`.)

Delta data may be collected at the meter with a measuring period built into the measurement. If so, this period should appear as attribute #3 in the ReadingType string. When this is the case, it is not required to send both the `<start>` and `<end>` tags in every message. The missing tag can be calculated. Given that only one tag is needed, and that the `<end>` tag is the absolute timestamp, then only the `<end>` tag need be supplied in this situation.

The other type of reading which might be periodically sampled is snapshot data. These are typically “indicating” values. If these are to be published along with a mix of other data types, and a common `<valuesInterval>` timestamp is provided for all of the readings, and only an

8 http://en.wikipedia.org/wiki/ISO_31-11

`<end>` tag is supplied, then this `<end>` tag could be said to apply to the indicating value just as it would to a delta data value.

If there is any ambiguity surrounding the timespans involved or the moment at which the reading was taken, then the `<MeterReading><Readings><timestamp>` element should be used liberally. When a timestamp is supplied in the `<MeterReading><Readings><timestamp>` element, and a different timestamp supplied in `<MeterReading><valuesInterval><end>`, the question arises which takes precedence? This is a contradiction and should not happen, but if it does, the timestamp closest to the value – that is, the `<MeterReading><Readings><timestamp>` – takes precedence over what the global default timestamp, `<MeterReading>` `<valuesInterval><end>` states.

6.7.3 The interval block

A system may publish interval data under the `<IntervalBlocks>` element or a series of ordinary `<Readings>` elements. The organization of the `<IntervalBlocks>` element provides an optimization where a common `<ReadingType>` may be pulled out and stated once for a long series of `<intervalReadings>`.

This optimization may be useful when the system publishes a series of intervals of the same type which span a considerable amount of time. It is probably less useful when the system publishes a number of readings with different `<ReadingType>` and a common timestamp.

6.7.4 Raw data

The `<IntervalBlocks>` elements are unique in that they allow raw data from the meter to be expressed along with a pending unit of measure. This is achieved by means of the `<PendingCalculation>` subelements.

This is useful in situations where delta data is transported and concerns arise over the loss of integrity because of rounding errors. By defering the mathematical operations until the data reaches the MDMS, roundoff errors within the HES can be avoided.

(It might be argued that with proper use of anchor readings, cumulative roundoff error could be avoided within the HES and the transport of raw data is unnecessary.)

Figure 39 shows an example message that illustrates usage of the `<IntervalBlocks>` elements.

```
<EventMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>created</Verb>
    <Noun>MeterReadings</Noun>
    <Timestamp>2013-06-26T17:35:00Z</Timestamp>
    <MessageID>23bbf080-bd7d-4897-a93f-815c8b56bc6d</MessageID>
    <CorrelationID>8c776dc7-83d1-4032-a2c5-6c4c1f06cf97</CorrelationID>
  </Header>
  <Payload>
    <MeterReadings xmlns="http://iec.ch/TC57/2011/MeterReadings#">
      <MeterReading>
        <IntervalBlocks>
          <IntervalReadings>
            <value>12345</value>
            <timePeriod>
              <end>2001-12-17T09:00:00Z</end>
              <start>2001-12-17T08:00:00Z</start>
            </timePeriod>
          </IntervalReadings>
          <PendingCalculation>
            <scalarDenominator>10000</scalarDenominator>
            <scalarNumerator>18</scalarNumerator>
            <ReadingType ref="0.0.7.4.1.1.12.0.0.0.0.0.0.0.0.3.72.0"/>
          <!--sixtyMinute   deltaData   forward   electricitySecondaryMetered   energy
(kWh)-->
          </PendingCalculation>
          <ReadingType ref="0.0.7.4.1.1.12.0.0.0.0.0.0.0.0.3.111.0"/>
        <!--sixtyMinute   deltaData   forward   electricitySecondaryMetered   energy
(count)-->
        </IntervalBlocks>
        <Meter>
          <Names>
            <name>meter1</name>
          </Names>
        </Meter>
      </MeterReading>
    </MeterReadings>
  </Payload>
</EventMessage>
```

Figure 39 – Example use of IntervalBlocks

The transport of raw data is not recommended but nevertheless supported. It should only be used in situations where systems cannot otherwise avoid cumulative round off error from occurring in the transport of delta data.

7 Meter control requests and responses

7.1 General

Meter control requests are used, among other things, to instruct the meter to reset itself, to connect the power or to disconnect the power.

A typical example of the usage of such requests is when someone moves into an apartment. The meter first needs to be read so that a final bill can be produced for the old tenant. It is then reset before the new tenant moves in.

Power disconnection may happen, for instance, as a last resort if accounts are not settled.

7.2 Message exchange pattern

From an IEC 61968-9 perspective, the message exchange patterns in these cases are all similar to one another and are depicted in Figure 40.

The steps are as follows:

1) The MDMS first sends a create(EndDeviceControls) message to the HES. This specifies a set of meter identifiers along with an EndDeviceControl code. An EndDeviceControl code is a four-position string that specifies the operation to be carried out. For example, the string `3.8.0.214` means "reset the meter". Meters are identified just as for meter read requests as described in 5.2 and 6.3.2. The EndDeviceControls code and the meter identifier are contained in the message `<Payload>` as shown in 7.2.
2) The HES checks the request message and sends a corresponding reply(EndDeviceControls) message back to the MDMS. This indicates whether the request looks acceptable or whether any errors have been detected. For example, the request may specify meter identifiers that do not exist.
3) The HES then endeavors to carry out the requests by sending the appropriate commands over the AMI network. When a meter receives a command, it attempts to obey it and then sends a notification of success or otherwise back to the HES.
4) The HES in turn notifies the MDMS by sending one or more created(EndDeviceEvents) messages. An EndDeviceEvents payload contains a set of meter identifiers together with an EndDeviceEvent code. The latter is a four-position string that signifies that some event or state change has taken place. For example, the string `3.8.0.215` denotes that the meter has been successfully reset.

A set of common EndDeviceControls and their corresponding EndDeviceEvents is shown in Table 1.

Table 1 – Common EndDeviceControls and their corresponding EndDeviceEvent codes

Operation	EndDeviceControls code	Expected EndDeviceEvent code	
		success case	failure case
Remote demand reset	3.8.0.214	3.8.0.215	3.8.0.65
Remote meter disconnect	3.31.0.23	3.31.0.68	3.31.0.84
Remote meter connect	3.31.0.18	3.31.0.42	3.31.0.67

A list of common EndDeviceEvents and an explanation of how these four-position strings are formed may be found in Annex E of IEC 61968-9:2013. A similar listing of common EndDeviceControls and an explanation of how these are constructed may be found in Annex F of IEC 61968-9:2013.

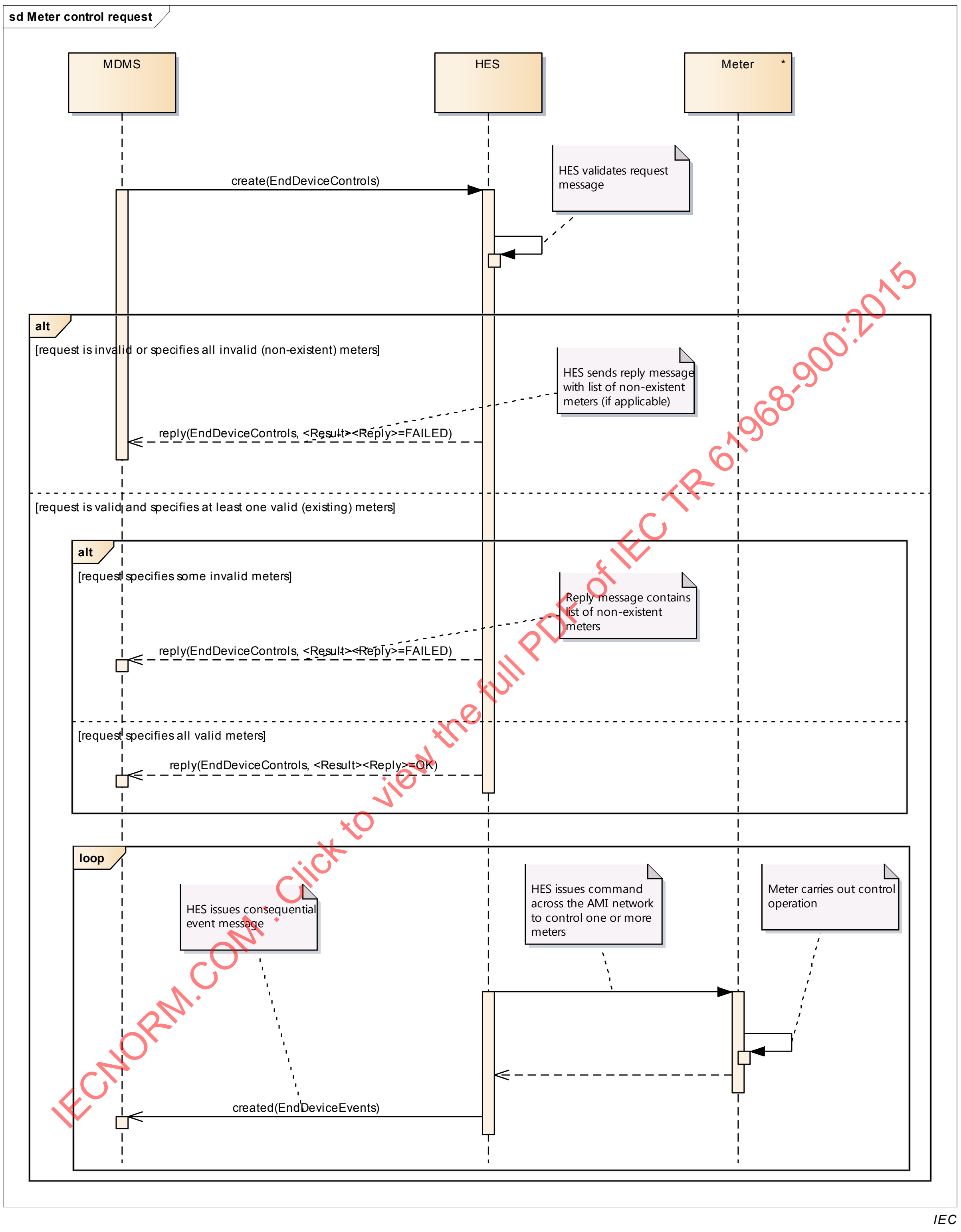

sd Meter control request
MDMS
HES
Meter *
HES validates request message
create(EndDeviceControls)
alt
[request is invalid or specifies all invalid (non-existent) meters]
HES sends reply message with list of non-existent meters (if applicable)
reply(EndDeviceControls, <Result><Reply>=FAILED)
[request is valid and specifies at least one valid (existing) meters]
alt
[request specifies some invalid meters]
Reply message contains list of non-existent meters
reply(EndDeviceControls, <Result><Reply>=FAILED)
[request specifies all valid meters]
reply(EndDeviceControls, <Result><Reply>=OK)
loop
HES issues consequential event message
HES issues command across the AMI network to control one or more meters
Meter carries out control operation
created(EndDeviceEvents)
IEC

Figure 40 – Message exchange pattern for a meter control operation

7.3 Create(EndDeviceControls) message

7.3.1 General

A simple create(EndDeviceControls) message that specifies a single meter might look as shown in Figure 41:

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>create</Verb>
    <Noun>EndDeviceControls</Noun>
    <Timestamp>2013-10-14T11:31:55Z</Timestamp>
    <ReplyAddress>http://localhost:8090/foobar</ReplyAddress>
    <MessageID>7c3bd408-c6df-4943-9ee7-38a326b83cca</MessageID>
    <CorrelationID>7c843d20-b47a-444a-90b3-3a23b6c52eae</CorrelationID>
  </Header>
  <Payload>
    <EndDeviceControls xmlns="http://iec.ch/TC57/2011/EndDeviceControls#">
      <EndDeviceControl>
        <EndDeviceControlType ref="3.8.0.214"/>
        <EndDevices>
          <Names>
            <name>M1001</name>
          </Names>
        </EndDevices>
      </EndDeviceControl>
    </EndDeviceControls>
  </Payload>
</RequestMessage>
```

Figure 41 – Example of a create(EndDeviceControls) message for one meter

A slightly more complicated example in which two meters are specified is shown in Figure 42:

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>create</Verb>
    <Noun>EndDeviceControls</Noun>
    <Timestamp>2013-10-14T11:31:55Z</Timestamp>
    <ReplyAddress>http://localhost:8090/foobar</ReplyAddress>
    <MessageID>7fb01d8b-383e-4642-877b-3ddc3a41cdcb</MessageID>
    <CorrelationID>806454a3-8ecb-46b5-a296-e0e2e3c9d8ea</CorrelationID>
  </Header>
  <Payload>
    <EndDeviceControls xmlns="http://iec.ch/TC57/2011/EndDeviceControls#">
      <EndDeviceControl>
        <EndDeviceControlType ref="3.8.0.214"/>
        <EndDevices>
          <Names>
            <name>M1001</name>
          </Names>
        </EndDevices>
        <EndDevices>
          <Names>
            <name>M1002</name>
          </Names>
        </EndDevices>
      </EndDeviceControl>
    </EndDeviceControls>
  </Payload>
</RequestMessage>
```

Figure 42 – Example of a create(EndDeviceControls) message for two meters

7.3.2 EndDeviceControls element

An `<EndDeviceControls>` element consists of zero or more `<EndDeviceControl>` subelements followed by zero or more `<EndDeviceControlType>` subelements. This is shown in Figure 43.

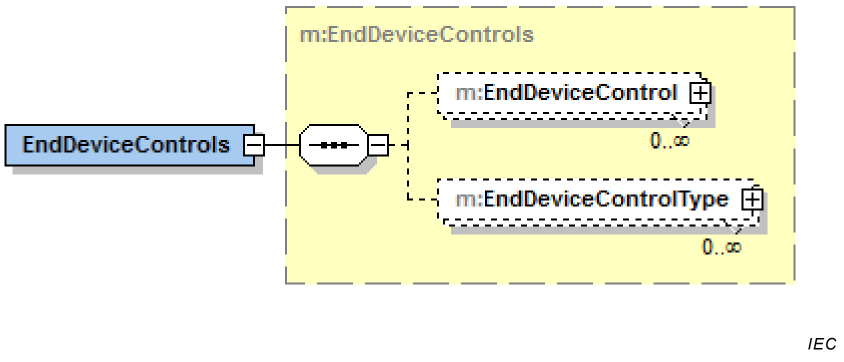
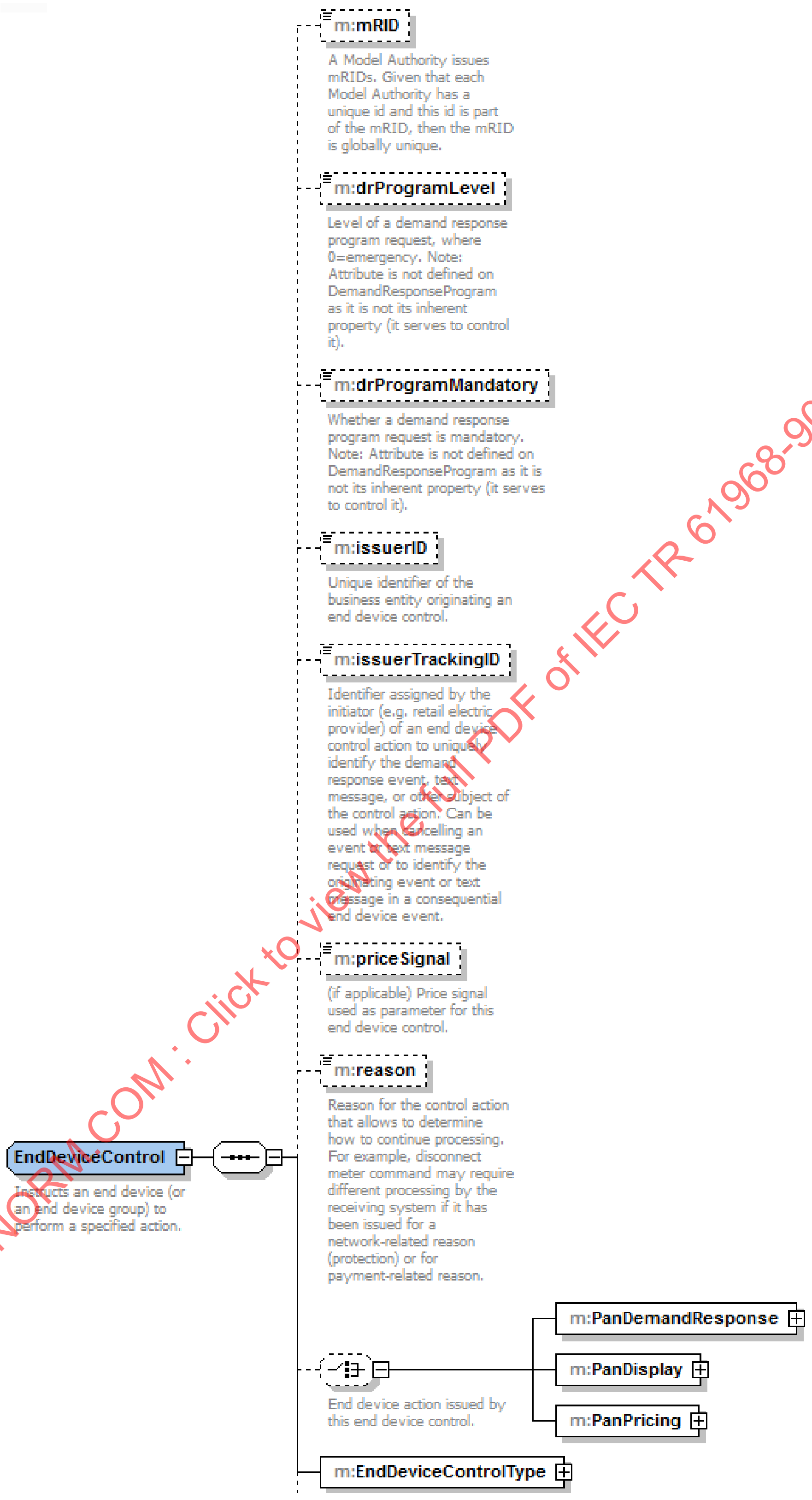


Figure 43 – EndDeviceControls definition

The `<EndDeviceControls><EndDeviceControl>` subelement contains at least an <EndDeviceControlType> which in turn contains a `ref=` attribute. It also contains `<EndDevices>`, `<EndDeviceGroups>`, `<UsagePoints>` and `<UsagePointGroups>` elements which denote the respective devices. This is depicted in Figure 44.


m:mRID
A Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part of the mRID, then the mRID is globally unique.
m:drProgramLevel
Level of a demand response program request, where 0=emergency. Note: Attribute is not defined on DemandResponseProgram as it is not its inherent property (it serves to control it).
m:drProgramMandatory
Whether a demand response program request is mandatory. Note: Attribute is not defined on DemandResponseProgram as it is not its inherent property (it serves to control it).
m:issuerID
Unique identifier of the business entity originating an end device control.
m:issuerTrackingID
Identifier assigned by the initiator (e.g. retail electric provider) of an end device control action to uniquely identify the demand response event, text message, or other subject of the control action. Can be used when cancelling an event or text message request or to identify the originating event or text message in a consequential end device event.
m:priceSignal
(if applicable) Price signal used as parameter for this end device control.
m:reason
Reason for the control action that allows to determine how to continue processing. For example, disconnect meter command may require different processing by the receiving system if it has been issued for a network-related reason (protection) or for payment-related reason.
EndDeviceControl
Instructs an end device (or an end device group) to perform a specified action.
m:PanDemandResponse
m:PanDisplay
m:PanPricing
End device action issued by this end device control.
m:EndDeviceControlType

IEC

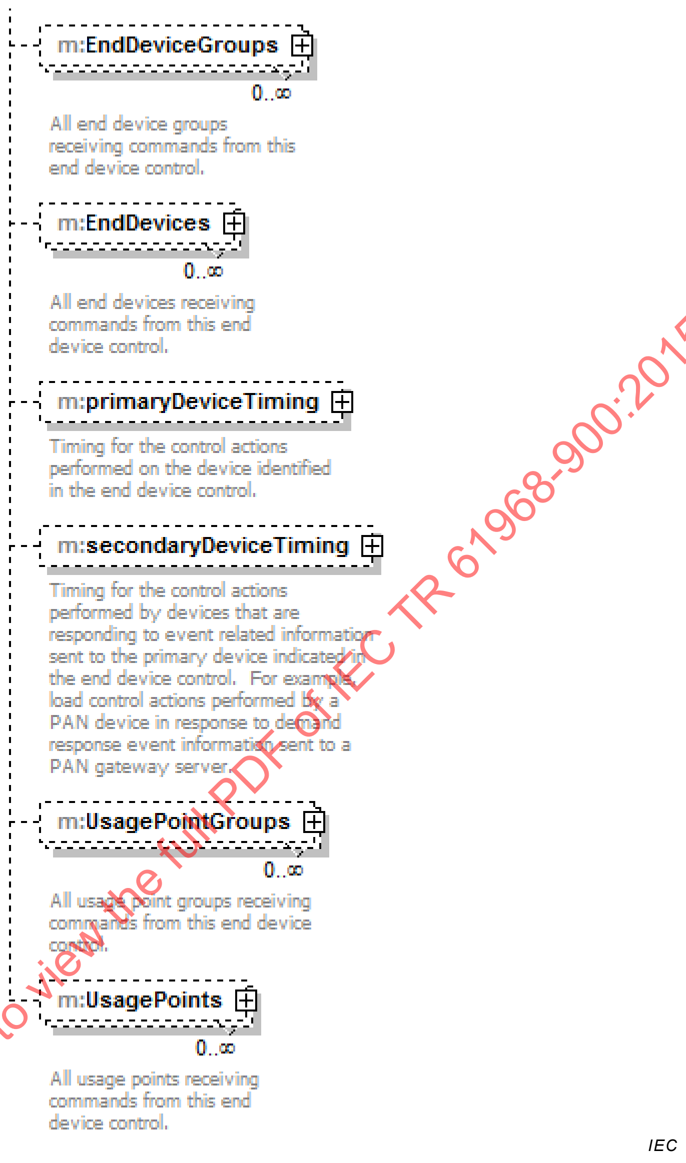

m:EndDeviceGroups
0..∞
All end device groups receiving commands from this end device control.
m:EndDevices
0..∞
All end devices receiving commands from this end device control.
m:primaryDeviceTiming
Timing for the control actions performed on the device identified in the end device control.
m:secondaryDeviceTiming
Timing for the control actions performed by devices that are responding to event related information sent to the primary device indicated in the end device control. For example, load control actions performed by a PAN device in response to demand response event information sent to a PAN gateway server.
m:UsagePointGroups
0..∞
All usage point groups receiving commands from this end device control.
m:UsagePoints
0..∞
All usage points receiving commands from this end device control.
IEC

Figure 44 – EndDeviceControl definition

The `<EndDeviceControls><EndDeviceControlType>` subelement may be used for associating a mnemonic name with an `<EndDeviceControlType>`, just as can be done for a MeterReadings response as described in 6.5.7. The corresponding XSD schema is shown in Figure 45.

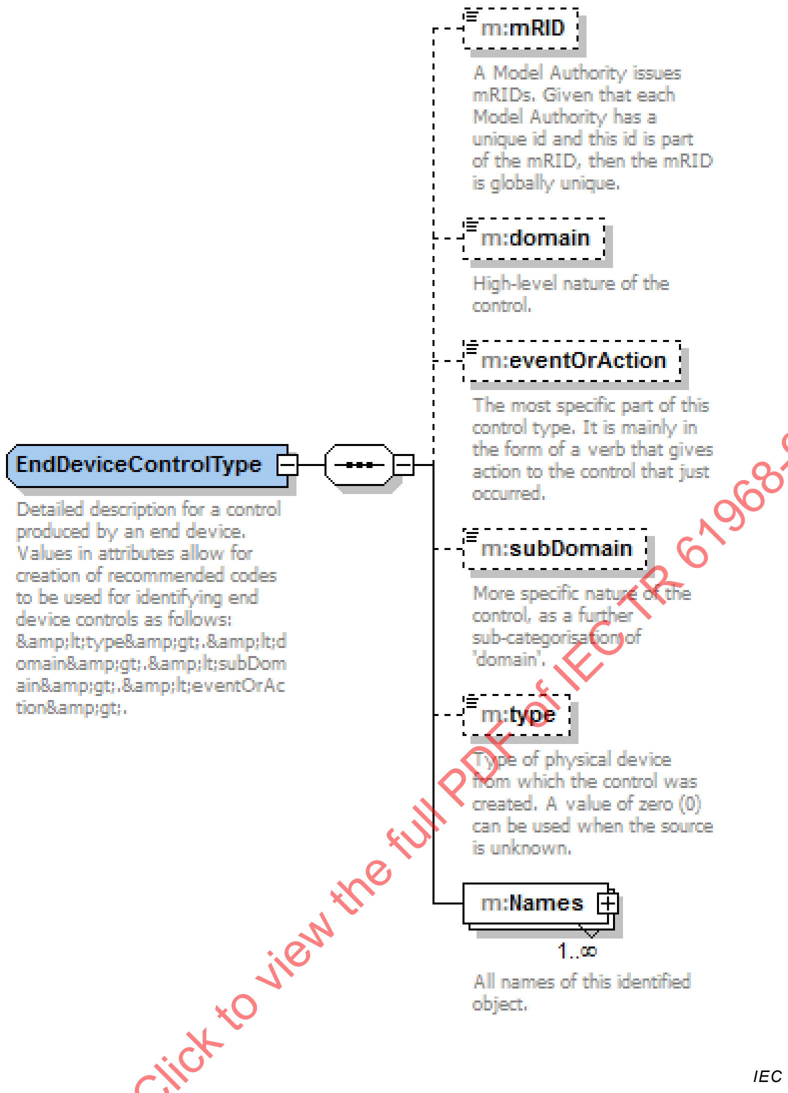

m:mRID
A Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part of the mRID, then the mRID is globally unique.
m:domain
High-level nature of the control.
m:eventOrAction
The most specific part of this control type. It is mainly in the form of a verb that gives action to the control that just occurred.
EndDeviceControlType
Detailed description for a control produced by an end device. Values in attributes allow for creation of recommended codes to be used for identifying end device controls as follows: &lt;type&gt;.&lt;domain&gt;.&lt;subDomain&gt;.&lt;eventOrAction&gt;.
m:subDomain
More specific nature of the control, as a further sub-categorisation of 'domain'.
m:type
Type of physical device from which the control was created. A value of zero (0) can be used when the source is unknown.
m:Names
1..∞
All names of this identified object.
IEC

Figure 45 – EndDeviceControlType definition

7.4 Reply(EndDeviceControls) message

A reply message corresponding to the create(EndDeviceControls) messages of Figure 42 is shown in Figure 46.

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>EndDeviceControls</Noun>
    <Timestamp>2013-10-14T11:31:57Z</Timestamp>
    <MessageID>6de7556b-b370-4040-9be5-1ce75838c6a4</MessageID>
    <CorrelationID>806454a3-8ecb-46b5-a296-e0e2e3c9d8ea</CorrelationID>
  </Header>
  <Reply>
    <Result>OK</Result>
    <Error>
      <code>0.0</code>
      <level>INFORM</level>
    </Error>
  </Reply>
</ResponseMessage>
```

Figure 46 – Example of a reply(EndDeviceControls) message

7.5 Created(EndDeviceEvents) message

7.5.1 General

Once the command has been sent over the AMI network to the meters and these have executed the commands, the HES responds back to the MDMS with a created(EndDeviceEvents) message as shown in Figure 47:

```
<EventMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>created</Verb>
    <Noun>EndDeviceEvents</Noun>
    <Timestamp>2013-10-14T11:31:57Z</Timestamp>
    <MessageID>6de7556b-b370-4040-9be5-1ce75838c6a4</MessageID>
    <CorrelationID>806454a3-8ecb-46b5-a296-e0e2e3c9d8ea</CorrelationID>
  </Header>
  <Payload>
    <EndDeviceEvents xmlns="http://iec.ch/TC57/2011/EndDeviceEvents#">
      <EndDeviceEvent>
        <createdDateTime>2013-10-14T11:31:57Z</createdDateTime>
        <EndDeviceEventType ref="3.8.0.215"/>
        <UsagePoint>
          <Names>
            <name>M1001</name>
          </Names>
        </UsagePoint>
      </EndDeviceEvent>
      <EndDeviceEvent>
        <createdDateTime>2013-10-14T11:31:57Z</createdDateTime>
        <EndDeviceEventType ref="3.8.0.215"/>
        <UsagePoint>
          <Names>
            <name>M1002</name>
          </Names>
        </UsagePoint>
      </EndDeviceEvent>
    </EndDeviceEvents>
  </Payload>
</EventMessage>
```

Figure 47 – Example of a created(EndDeviceEvents) message

7.5.2 EndDeviceEvents element

An <EndDeviceEvents> element consists of zero or more <EndDeviceEvent> subelements followed by zero or more <EndDeviceEventType> subelements. This is shown in Figure 48.

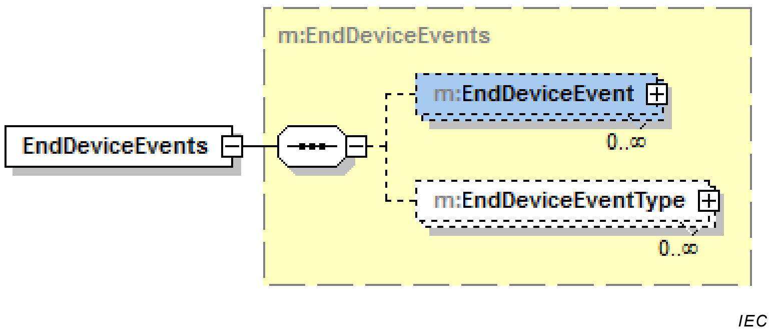
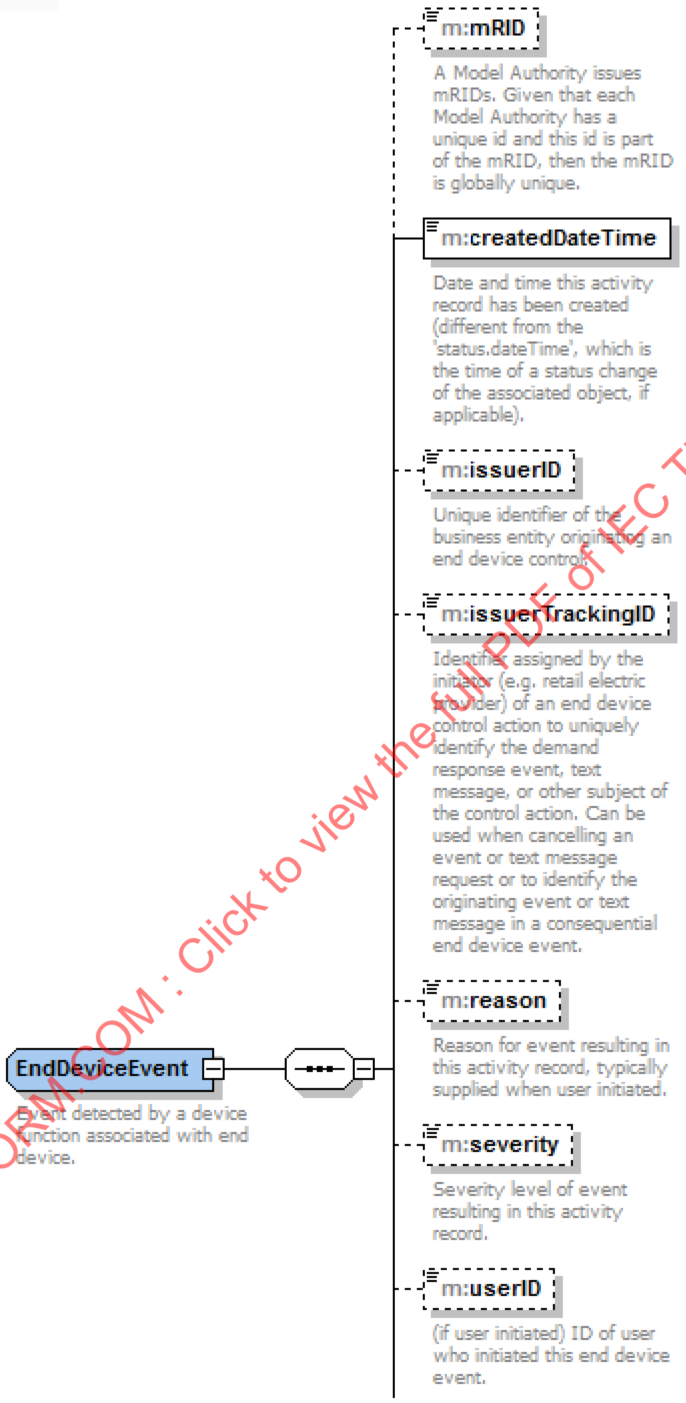


Figure 48 – EndDeviceEvents definition

The `<EndDeviceEvents><EndDeviceEvent>` subelement contains at least an `<EndDeviceEventType>` which in turn contains a `ref=` attribute. It also contains `<Names>` and `<UsagePoints>` subelements which denote the respective devices. This is shown in Figure 49.


m:mRID
A Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part of the mRID, then the mRID is globally unique.
m:createdDateTime
Date and time this activity record has been created (different from the 'status.dateTime', which is the time of a status change of the associated object, if applicable).
m:issuerID
Unique identifier of the business entity originating an end device control.
m:issuerTrackingID
Identifier assigned by the initiator (e.g. retail electric provider) of an end device control action to uniquely identify the demand response event, text message, or other subject of the control action. Can be used when cancelling an event or text message request or to identify the originating event or text message in a consequential end device event.
m:reason
Reason for event resulting in this activity record, typically supplied when user initiated.
m:severity
Severity level of event resulting in this activity record.
m:userID
(if user initiated) ID of user who initiated this end device event.
EndDeviceEvent
Event detected by a device function associated with end device.

IEC

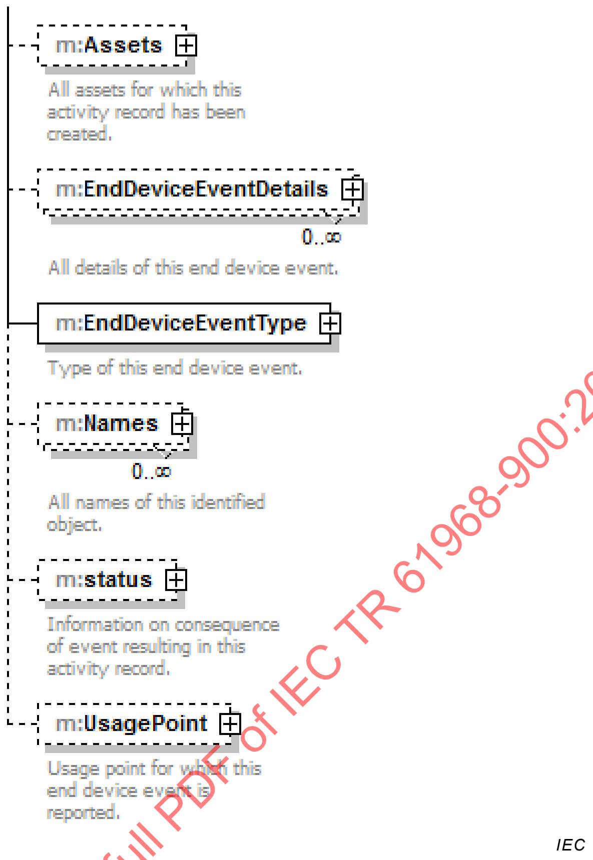

m:Assets
All assets for which this activity record has been created.
m:EndDeviceEventDetails
0..∞
All details of this end device event.
m:EndDeviceEventType
Type of this end device event.
m:Names
0..∞
All names of this identified object.
m:status
Information on consequence of event resulting in this activity record.
m:UsagePoint
Usage point for which this end device event is reported.
IEC

Figure 49 – EndDeviceEvent definition

The `<EndDeviceEvents><EndDeviceEventType>` subelement may be used for associating a mnemonic name with an `<EndDeviceEventType>`, just as can be done for a MeterReadings and EndDeviceControls profiles as described in 6.5.7 and 7.3.2.

Figure 50 depicts the structure of the `<EndDeviceEvents><EndDeviceEventType>` subelement.

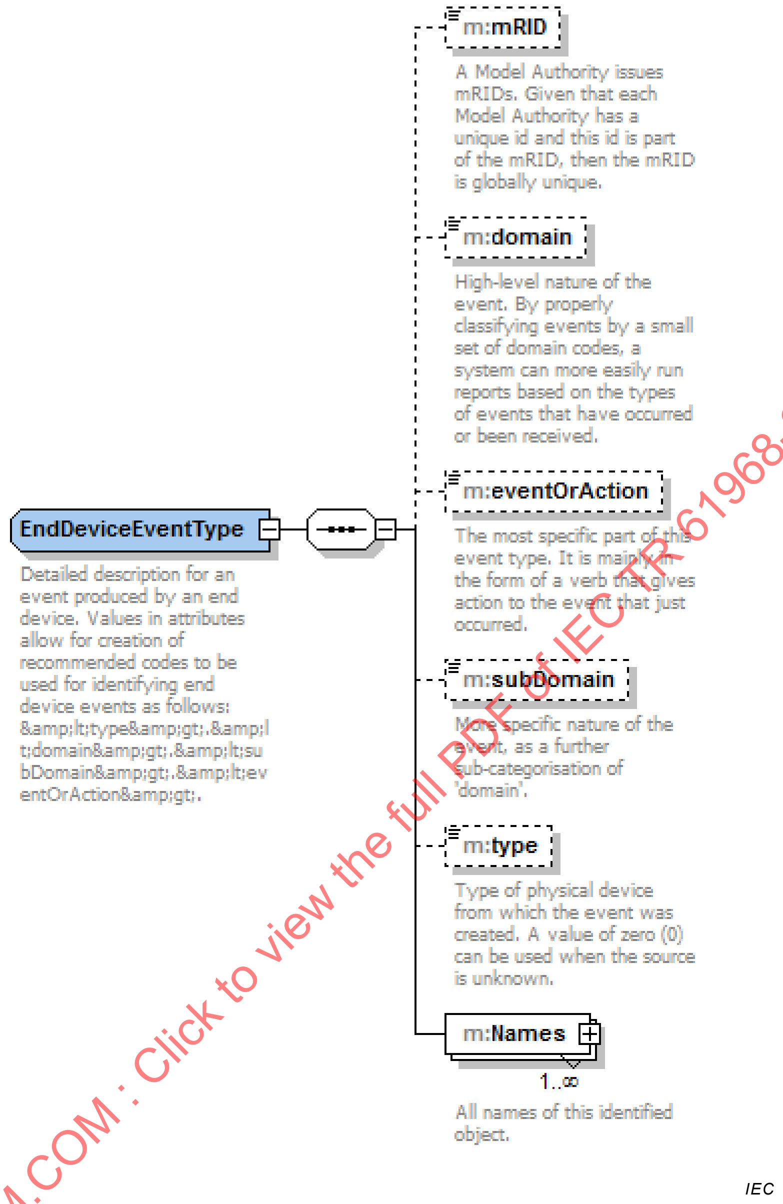

EndDeviceEventType
Detailed description for an event produced by an end device. Values in attributes allow for creation of recommended codes to be used for identifying end device events as follows: &lt;type&gt;.&lt;domain&gt;.&lt;subDomain&gt;.&lt;eventOrAction&gt;.
m:mRID
A Model Authority issues mRIDs. Given that each Model Authority has a unique id and this id is part of the mRID, then the mRID is globally unique.
m:domain
High-level nature of the event. By properly classifying events by a small set of domain codes, a system can more easily run reports based on the types of events that have occurred or been received.
m:eventOrAction
The most specific part of this event type. It is mainly in the form of a verb that gives action to the event that just occurred.
m:subDomain
More specific nature of the event, as a further sub-categorisation of 'domain'.
m:type
Type of physical device from which the event was created. A value of zero (0) can be used when the source is unknown.
m:Names
1..∞
All names of this identified object.
IEC

Figure 50 – EndDeviceEventType definition

7.6 Unsolicited EndDeviceEvents Messages

7.6.1 General

An unsolicited created(EndDeviceEvents) message may be sent from a HES to a MDMS to inform the latter of some unexpected event. For example, the HES may notify the MDMS of a power outage.

Table 2 shows a list of common EndDeviceEvents. Further examples are to be found in Annex E of IEC 61968-9:2013.

Table 2 – Common EndDeviceEvent codes for unsolicited messages

Operation	EndDeviceEvent code
Power outage	3.26.0.85
Tamper violation	3.12.0.257
High temperature alert	3.35.0.40
Low voltage alarm	3.26.38.150
High voltage alarm	3.26.38.93

7.6.2 Message exchange pattern

In diagrammatic terms the message exchange pattern is as shown in Figure 51:

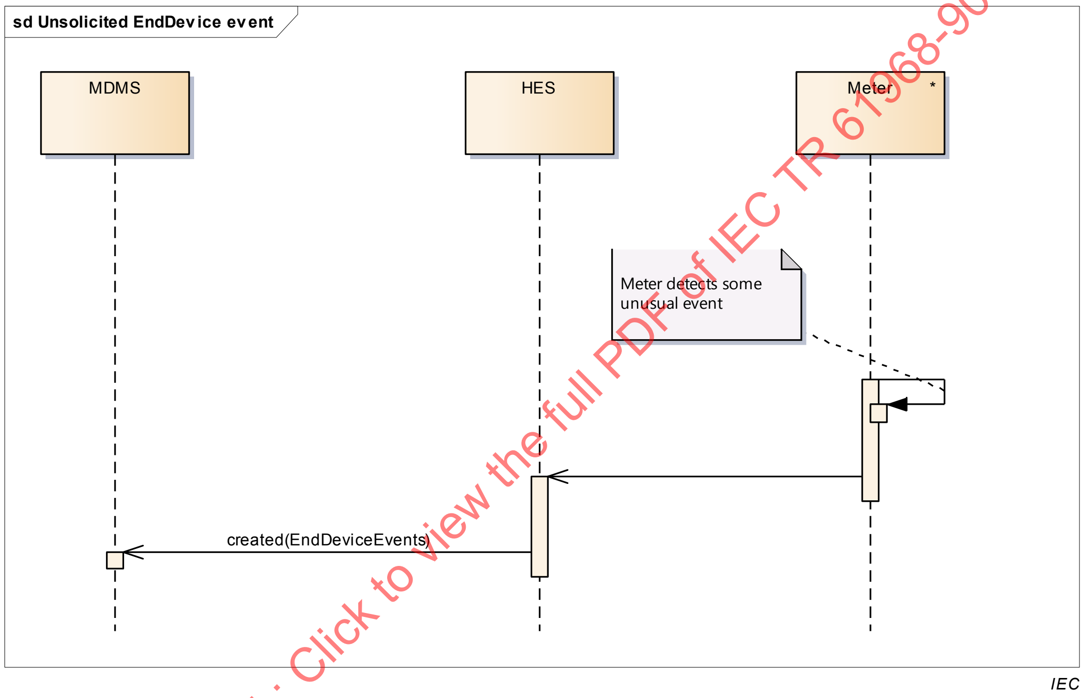

sd Unsolicited EndDevice event
MDMS
HES
Meter
*
Meter detects some unusual event
created(EndDeviceEvents)
IEC

Figure 51 – Message exchange pattern for an unsolicted EndDevice event

7.7 Premises area networks

7.7.1 General

From the perspective of IEC 61968-9, a premises area network (PAN) comprises a set of devices attached to a meter. Such PAN devices typically include home displays, thermostats, relays and so on. The meter acts as a gateway between the AMI network and the PAN devices.

See Subclause 5.9 of IEC 61968-9:2013 for a full discussion of PAN networks.

7.7.2 Message exchange pattern

Before any command can be sent to a PAN device it must first be attached to – or paired with – the meter. The pairing procedure and the subsequent control operations on the PAN device are defined in the same way as other meter control messages.

That is, the sequence of events is initiated by the MDMS sending a create(EndDeviceControls) message to the HES. The HES sends a corresponding command over the AMI network to the meter which then responds with a reply(EndDeviceControls) message. The meter then issues the requisite command over the PAN network to the PAN device. The PAN device responds back to the meter which in turn sends a reponse message back to the HES. The latter then sends a created(EndDeviceEvents) message to the MDMS.

The major difference therefore between PAN operations and other meter control operations lies in the choice of EndDeviceControls and EndDeviceEvents codes. The relevant codes are listed in Tables F.3 and E.24 of IEC 61968-9:2013 respectively.

Another difference between PAN operations and other meter control operations is that the EndDeviceControls element allows for PAN-specific operations. For instance, the `<EndDeviceControl><priceSignal>` element allows price information to be incorporated. Similarly, the `<EndDeviceControl><PANDisplay>` elements allows for text messages to be displayed on home PAN device. Finally the `<EndDeviceControl><PANDemandResponse>` element allows for dynamic load-control information to be sent to a PAN device.

7.7.3 Pairing the Meter and PAN device

The pairing operation is accomplished by sending a “PAN pairing window open” EndDeviceControls message to the meter. The corresponding EndDeviceControls code is `3.10.73.298` assuming an electricity meter. The expected EndDeviceEvents code for the success case is `3.10.73.39`.

This message should specify two EndDevices, one of which (the PAN device) has the `<EndDeviceControls><EndDeviceControl><EndDevices><isPan>` element set to `true` and the other (the meter) omits this element or sets its value to `false`. For the former, the `<EndDeviceControls><EndDeviceControl><EndDevices><installCode>` and `<EndDeviceControls><EndDeviceControl><EndDevices><electronicAddress><mac>` elements must typically also be set.

Once the pairing operation has been accomplished, a “PAN pairing window close” EndDeviceControls message can be sent to the meter. The corresponding EndDeviceControls and expected EndDeviceEvents codes are `3.10.73.299` and `3.10.73.16` respectively.

8 Configuration and provisioning

8.1 General

A MDMS or a HES may be configured with information concerning entities such as meters, customers, customer accounts, service suppliers, usage points and so on. They can also be configured with information defining the relationships or associations among such entities.

IEC 61968-9 defines several kinds of configuration according to the kind of entity being provisioned. It also defines how this configuration information can be added, changed or deleted.

The different kinds of configuration information are shown in Table 3.

Table 3 – Config Profiles

Profile (XSD)	See Clause
MeterConfig	8.3, 11.4.2
ComModuleConfig	11.4.3
ServiceLocationConfig	11.4.4
ServiceCategoryConfig	11.4.5
ServiceSupplierConfig	11.4.6
UsagePointLocationConfig	11.4.7
UsagePointConfig	11.4.8
CustomerConfig	11.4.9
CustomerAccountConfig	11.4.10
CustomerAgreementConfig	11.4.11
PricingStructureConfig	11.4.12
MasterDataLinkageConfig *(manage associations between objects)*	8.4, 11.4.13
ObjectNamesConfig *(change the identifiers of objects)*	11.4.15

These Config profiles are used to add, delete or modify the corresponding configuration information. For instance, to provision a HES with a new meter, a message is sent formulated according to the `MeterConfig` profile. Similary, to inform a HES of the existence of a new customer, the `CustomerConfig` profile is used.

8.2 Message exchange pattern

The message exchange patterns are the same in all cases. To add configuration data to a HES or other system, for example, a request message containing a header with a `create` verb is sent. The configuration data is contained within the message `<Payload>` element. Once the operation has been carried out, the response message is returned with a verb of `reply` and an indication of the success or otherwise of the request in the message `<Reply>` element.

Requests to delete or modify existing configuration data function similarly except that the header verbs are `delete` and `change` respectively.

Depending on the scenario, a given create, change, or delete request may cause several reply messages to be generated. For example, a single create message can be issued to instantiate multiple Meters. In this case, the responding system can send a single reply message for all Meters or multiple reply messages with the reply data with just acknowledgements for one or more Meters in each message.

Figure 52 illustrates an example message exchange pattern for the case when a meter is being provisioned into the system. The same general message pattern is also used for nouns other than `MeterConfig`. Similarly, the verbs `change` and `delete` may be used instead of `create` for when the stored configuration information is being updated or removed.

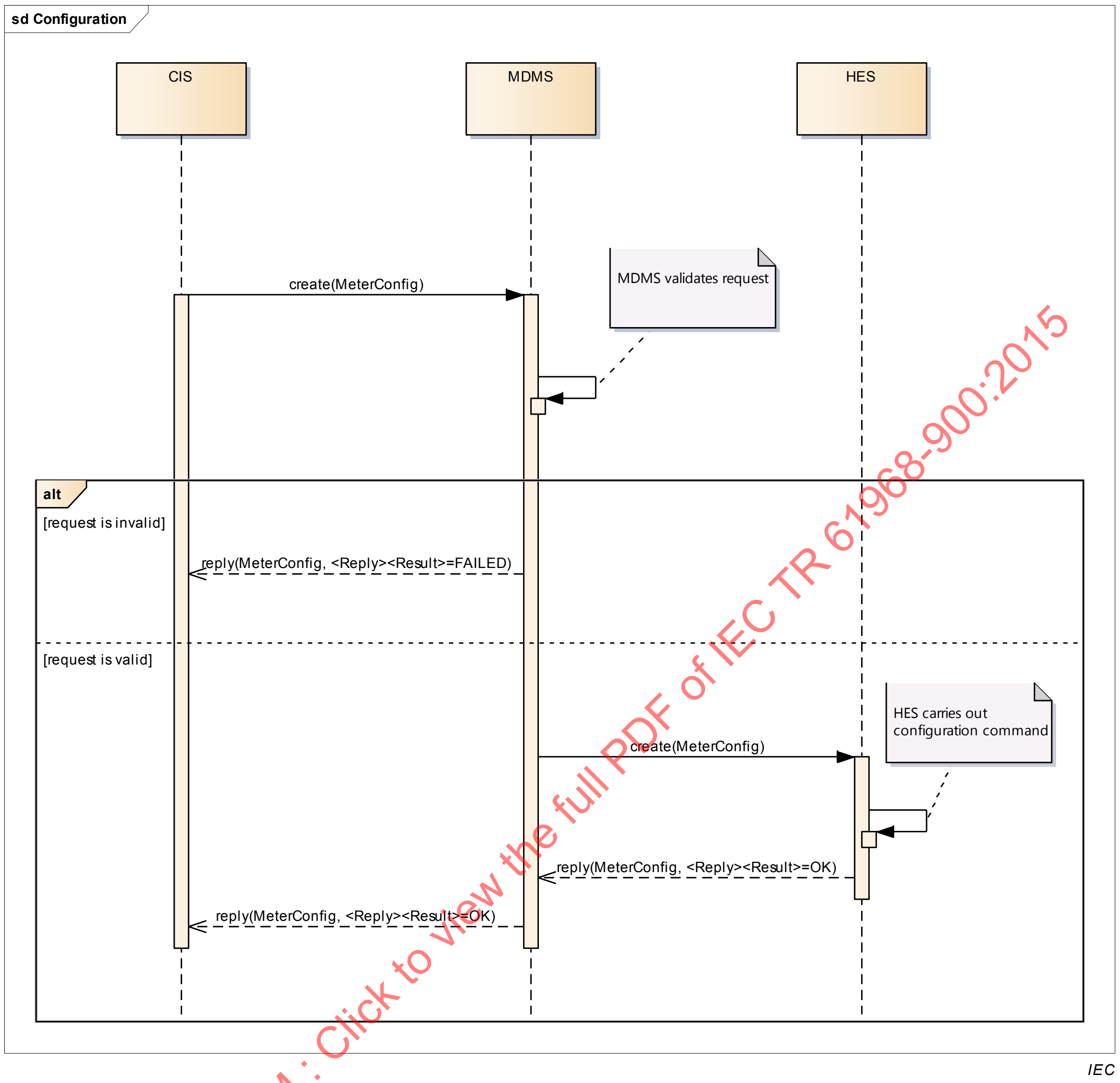

sd Configuration
CIS
MDMS
HES
create(MeterConfig)
MDMS validates request
alt
[request is invalid]
reply(MeterConfig, <Reply><Result>=FAILED)
[request is valid]
HES carries out configuration command
create(MeterConfig)
reply(MeterConfig, <Reply><Result>=OK)
reply(MeterConfig, <Reply><Result>=OK)
IEC

Figure 52 – Message exchange pattern for a create(MeterConfig) message

8.3 Meter configuration

8.3.1 General

Figure 53 shows the various parameters that can be passed in a MeterConfig request.

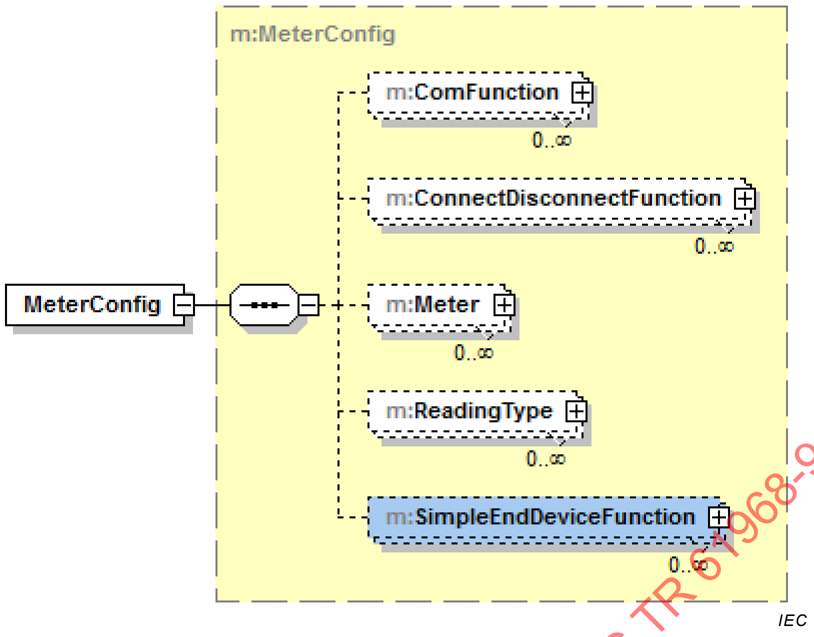


Figure 53 – MeterConfig definition

See 11.4.2 for a detailed description of the various fields in such a message.

8.3.2 Create(MeterConfig) message

The example shown in Figure 54 is taken from Annex L of IEC 61968-9:2013. It shows a sample request message for the provisioning of two meters.

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>create</Verb>
    <Noun>MeterConfig</Noun>
    <Timestamp>2012-01-17T09:30:47Z</Timestamp>
    <Source>CIS</Source>
    <AckRequired>true</AckRequired>
    <MessageID>8B3EF3E8-C61C-4C91-BEF0-A1775570656A</MessageID>
    <CorrelationID>8B3EF3E8-C61C-4C91-BEF0-A1775570656A</CorrelationID>
  </Header>
  <Payload>
    <MeterConfig xmlns="http://iec.ch/TC57/2011/MeterConfig#">
      <ComFunction>
        <mRID>14470EB8-F28F-433E-9A34-D61C845FD976</mRID>
        <direction>biDirectional</direction>
        <technology>cellular</technology>
      </ComFunction>
      <Meter>
        <mRID>B95ED625-2EDB-437F-977C-6E2991EE61CB</mRID>
        <amrSystem>AmrSystemName</amrSystem>
        <formNumber>2S</formNumber>
        <ConfigurationEvents>
          <createdDateTime>
            2001-12-17T09:30:47Z</createdDateTime>
          <effectiveDateTime>
            2001-12-19T00:00:00Z</effectiveDateTime>
        </ConfigurationEvents>
        <electronicAddress>
          <mac>00:24:E8:A7:69:E7</mac>
        </electronicAddress>
        <ComFunction ref="14470EB8-F28F-433E-9A34-D61C845FD976"/>
        <EndDeviceInfo>
          <isSolidState>true</isSolidState>
          <phaseCount>1</phaseCount>
          <capability>
            <autonomousDst>true</autonomousDst>
            <communication>true</communication>
            <connectDisconnect>false</connectDisconnect>
            <electricMetering>true</electricMetering>
            <metrology>true</metrology>
            <onRequestRead>true</onRequestRead>
          </capability>
        </EndDeviceInfo>
        <Names>
          <name>A47129</name>
          <NameType>
            <name>MeterBadgeNumber</name>
            <NameTypeAuthority>
              <name>UtilityXYZ</name>
            </NameTypeAuthority>
          </NameType>
        </Names>
      </Meter>
      <Meter>
        <mRID>ED04579E-FF0F-418E-92AC-B8BBE5721870</mRID>
        <amrSystem>AmrSystemName</amrSystem>
        <formNumber>2S</formNumber>
        <ConfigurationEvents>
          <createdDateTime>2001-12-17T09:30:47Z</createdDateTime>
          <effectiveDateTime>2001-12-19T00:00:00Z</effectiveDateTime>
        </ConfigurationEvents>
        <electronicAddress>
          <mac>00:24:E8:B7:79:E7</mac>
        </electronicAddress>
        <ComFunction ref="14470EB8-F28F-433E-9A34-D61C845FD976"/>
        <EndDeviceInfo>
          <isSolidState>true</isSolidState>
          <phaseCount>1</phaseCount>
          <capability>
            <autonomousDst>true</autonomousDst>
            <communication>true</communication>
            <connectDisconnect>false</connectDisconnect>
            <electricMetering>true</electricMetering>
            <metrology>true</metrology>
            <onRequestRead>true</onRequestRead>
          </capability>
        </EndDeviceInfo>
        <Names>
          <name>C57129</name>
          <NameType>
            <name>MeterBadgeNumber</name>
            <NameTypeAuthority>
              <name>UtilityXYZ</name>
            </NameTypeAuthority>
          </NameType>
        </Names>
      </Meter>
    </MeterConfig>
  </Payload>
</RequestMessage>
```

Figure 54 – Example of a create(MeterConfig) message

8.3.3 Reply(MeterConfig) message, success case

Assuming that the provisioning succeeds, the response message looks as shown in Figure 55:

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>MeterConfig</Noun>
    <Timestamp>2012-01-17T09:33:47Z</Timestamp>
    <Source>MDMS</Source>
    <MessageID>5624858B-9365-482E-8335-746A9A06F3FB</MessageID>
    <CorrelationID>8B3EF3E8-C61C-4C91-BEF0-A1775570656A</CorrelationID>
  </Header>
  <Reply>
    <Result>OK</Result>
    <Error>
      <code>0.0</code>
    </Error>
  </Reply>
</ResponseMessage>
```

Figure 55 – Example of a reply(MeterConfig) message, success case

8.3.4 Reply(MeterConfig) message, failure case

In the error case, the response message is as shown in Figure 56:

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>MeterConfig</Noun>
    <Timestamp>2012-01-17T09:33:47Z</Timestamp>
    <Source>MDMS</Source>
    <MessageID>5624858B-9365-482E-8335-746A9A06F3FB</MessageID>
    <CorrelationID>8B3EF3E8-C61C-4C91-BEF0-A1775570656A</CorrelationID>
  </Header>
  <Reply>
    <Error>
      <code>2.4</code>
      <level>FATAL</level>
      <details>Meter ID already exists</details>
      <ID idType="MeterBadgeNumber" idAuthority="UtilityXYZ"
          kind="name" objectType="Meter">C57129</ID>
    </Error>
  </Reply>
</ResponseMessage>
```

Figure 56 – Example of a reply(MeterConfig) message, failure case

8.4 Master data linkage

8.4.1 General

It is often necessary to establish an association between previously defined configuration entities. For example, a given meter may be associated with a particular usage point. Alternatively, a customer may be associated with a customer account or with a customer agreement.

Such bindings may sometimes be achieved by defining the appropriate fields in a `UsagePointConfig` profile. A more general way, however, of defining an association between different configuration entities is by means of the `MasterDataLinkage` profile.

The diagram shown in Figure 57 and taken from Subclause 5.10.2 of IEC 61968-9:2013 illustrates which associations may be made with the `UsagePointConfig` or `CustomerAgreementConfig` profiles and which should be established using the `MasterDataLinkage` profile

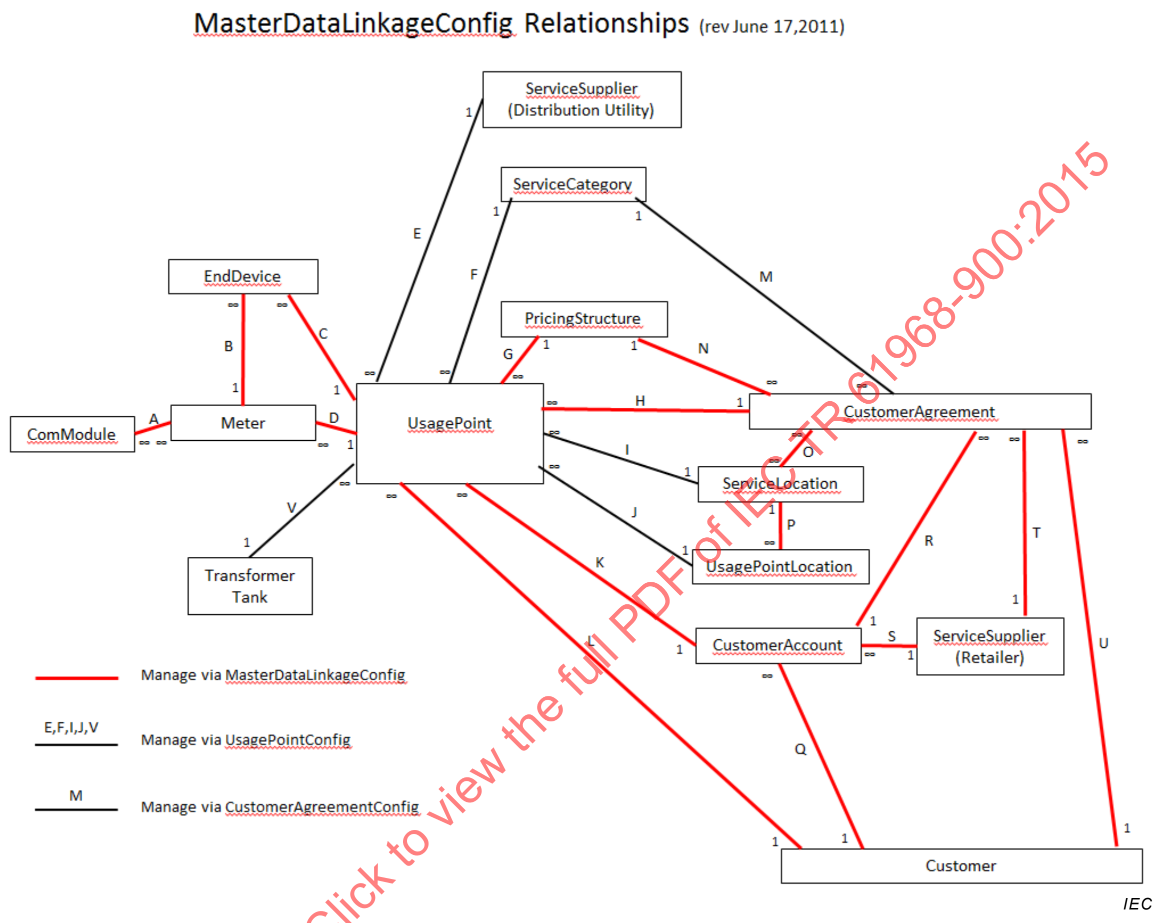

MasterDataLinkageConfig Relationships (rev June 17,2011)
ServiceSupplier (Distribution Utility)
ServiceCategory
EndDevice
PricingStructure
ComModule
Meter
UsagePoint
CustomerAgreement
ServiceLocation
UsagePointLocation
Transformer Tank
CustomerAccount
ServiceSupplier (Retailer)
Customer
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
Manage via MasterDataLinkageConfig
E,F,I,J,V
Manage via UsagePointConfig
M
Manage via CustomerAgreementConfig
IEC

Figure 57 – MasterDataLinkageConfig Relationships

A MasterDataLinkage request contains a header with a verb of `create` and a noun of `MasterDataLinkage`. The request element contains a MasterDataLinkage profile whose elements refer to previously defined configuration entities.

8.4.2 Create(MasterDataLinkage) message

The example message shown in Figure 58 shows a MasterDataLinkage request being used to set up an association between a usage point and a meter (line D in Figure 57):

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>create</Verb>
    <Noun>MasterDataLinkageConfig</Noun>
    <Timestamp>2012-01-17T10:00:47Z</Timestamp>
    <AckRequired>true</AckRequired>
    <MessageID>F47F3703-D16A-4D3C-901A-553E1E26EA03</MessageID>
    <CorrelationID>F47F3703-D16A-4D3C-901A-553E1E26EA03</CorrelationID>
  </Header>
  <Payload>
    <MasterDataLinkageConfig
xmlns="http://iec.ch/TC57/2011/MasterDataLinkageConfig#">
      <ConfigurationEvent>
        <createdDateTime>2012-01-17T10:00:47</createdDateTime>
        <effectiveDateTime>2012-12-15T10:30:47</effectiveDateTime>
      </ConfigurationEvent>
      <Meter>
        <Names>
          <name>A47129</name>
          <NameType>
            <name>MeterBadgeNumber</name>
            <NameTypeAuthority>
              <name>UtilityXYZ</name>
            </NameTypeAuthority>
          </NameType>
        </Names>
      </Meter>
      <UsagePoint>
        <Names>
          <name>UP124179</name>
          <NameType>
            <name>ServiceDeliveryPointID</name>
            <NameTypeAuthority>
              <name>UtilityXYZ</name>
            </NameTypeAuthority>
          </NameType>
        </Names>
      </UsagePoint>
    </MasterDataLinkageConfig>
  </Payload>
</RequestMessage>
```

Figure 58 – Example of a create(MasterDataLinkageConfig) message

8.5 OperationSets

8.5.1 General

An <OperationSet> is used to group several configuration control operations into a single transaction.

An OperationSet request contains a message header whose verb is execute and whose noun is OperationSet.

If the <OperationSet><enforceTransactionalIntegrity> element is true, then either all the operations succeed or they all fail. It is not possible for only some of them to succeed.

If the <OperationSet><enforceMsgSequence> element is true, then the operations are carried out strictly in the order defined by the <OperationSet><Operation><operationId> elements.

8.5.2 OperationSet request message

The <PayLoad> element contains an <OperationSet> element which in turn contains a set of <Operation> subelements. Each one of these may be thought of as its own self-contained payload. It contains a <noun>,<verb> and associated data.

The example given in Figure 59, taken from L.2.29 of IEC 61968-9:2013, makes this clear. This OperationSet first creates a Meter, then creates a UsagePoint, and then links the Meter and the UsagePoint, thereby reflecting the sequence of events that constitute a meter installation. In essence, it performs the same functions as several of the previous use cases, but does so in a single message and as a single transaction.

```
<RequestMessage
      xmlns="http://iec.ch/TC57/2011/schema/message"
      xmlns:m="http://iec.ch/TC57/2011/MeterConfig#"
      xmlns:up="http://iec.ch/TC57/2011/UsagePointConfig#"
      xmlns:mdlc="http://iec.ch/TC57/2011/MasterDataLinkageConfig#"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://iec.ch/TC57/2011/schema/message Message.xsd">
  <Header>
    <Verb>execute</Verb>
    <Noun>OperationSet</Noun>
    <Timestamp>2012-12-20T09:30:47Z</Timestamp>
    <Source>CIS</Source>
    <AckRequired>true</AckRequired>
    <MessageID>D921A053-80C1-4DB6-960E-2603127B7B92</MessageID>
    <CorrelationID>D921A053-80C1-4DB6-960E-2603127B7B92</CorrelationID>
  </Header>
  <Payload>
    <OperationSet>
      <enforceMsgSequence>true</enforceMsgSequence>
      <enforceTransactionalIntegrity>true</enforceTransactionalIntegrity>
      <Operation>
        <operationId>1</operationId>
        <noun>MeterConfig</noun>
        <verb>create</verb>
        <mdlc:MeterConfig>
          <mdlc:Meter>
            <mdlc:formNumber>2S</mdlc:formNumber>
            <mdlc:ConfigurationEvents>
            <mdlc:createdDateTime>2012-12-20T09:30:47Z</mdlc:createdDateTime>
          <mdlc:effectiveDateTime>2012-12-21T00:00:00Z</mdlc:effectiveDateTime>
            <mdlc:Names>
              <mdlc:name>C34531</mdlc:name>
              <mdlc:NameType>
                <mdlc:name>MeterBadgeNumber</mdlc:name>
                <mdlc:NameTypeAuthority>
                  <mdlc:name>UtilityXYZ</mdlc:name>
                </mdlc:NameTypeAuthority>
              </mdlc:NameType>
            </mdlc:Names>
          </mdlc:ConfigurationEvents>
          </mdlc:Meter>
        </mdlc:MeterConfig>
      </Operation>
      <Operation>
        <operationId>2</operationId>
        <noun>UsagePointConfig</noun>
        <verb>create</verb>
        <up:UsagePointConfig>
          <up:UsagePoint>
```

```
            <up:amiBillingReady>amiCapable</up:amiBillingReady>
            <up:connectionState>connected</up:connectionState>
            <up:isSdp>true</up:isSdp>
            <up:isVirtual>false</up:isVirtual>
            <up:phaseCode>B</up:phaseCode>
            <up:readCycle>ReadCycleJ</up:readCycle>
            <up:ConfigurationEvents>
              <up:createdDateTime>2012-12-20T09:30:47Z</up:createdDateTime>
              <up:effectiveDateTime>2012-12-21T00:00:00Z</up:effectiveDateTime>
            </up:ConfigurationEvents>
            <up:Names>
              <up:name>UP43639</up:name>
              <up:NameType>
                <up:name>ServiceDeliveryPointID</up:name>
                <up:NameTypeAuthority>
                  <up:name>UtilityXYZ</up:name>
                </up:NameTypeAuthority>
              </up:NameType>
            </up:Names>
          </up:UsagePoint>
        </up:UsagePointConfig>
      </Operation>
      <Operation>
        <operationId>3</operationId>
        <noun>MasterDataLinkageConfig</noun>
        <verb>create</verb>
        <mdlc:MasterDataLinkageConfig>
          <mdlc:ConfigurationEvent>
            <mdlc:createdDateTime>2012-12-17T09:30:47Z</mdlc:createdDateTime>
          <mdlc:effectiveDateTime>2012-12-21T00:00:00Z</mdlc:effectiveDateTime>
          </mdlc:ConfigurationEvent>
          <mdlc:Meter>
            <mdlc:Names>
              <mdlc:name>C34531</mdlc:name>
              <mdlc:NameType>
                <mdlc:name>MeterBadgeNumber</mdlc:name>
                <mdlc:NameTypeAuthority>
                  <mdlc:name>UtilityXYZ</mdlc:name>
                </mdlc:NameTypeAuthority>
              </mdlc:NameType>
            </mdlc:Names>
          </mdlc:Meter>
          <mdlc:UsagePoint>
            <mdlc:Names>
              <mdlc:name>UP43639</mdlc:name>
              <mdlc:NameType>
                <mdlc:name>ServiceDeliveryPointID</mdlc:name>
                <mdlc:NameTypeAuthority>
                  <mdlc:name>UtilityXYZ</mdlc:name>
                </mdlc:NameTypeAuthority>
              </mdlc:NameType>
            </mdlc:Names>
          </mdlc:UsagePoint>
        </mdlc:MasterDataLinkageConfig>
      </Operation>
    </OperationSet>
  </Payload>
</RequestMessage>
```

Figure 59 – Example of an execute(OperationSet) message

8.5.3 OperationSet Response Message

The corresponding reply in the success case is shown in Figure 60:

```
<ResponseMessage xmlns = "http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>OperationSet</Noun>
    <Timestamp>2012-12-20T09:30:49Z</Timestamp>
    <Source>MDMS</Source>
    <MessageID>A8E8FE70-6960-4724-BF1C-CE2804EEA633</MessageID>
    <CorrelationID>D921A053-80C1-4DB6-960E-2603127B7B92</CorrelationID>
  </Header>
  <Reply>
    <Result>OK</Result>
    <Error>
      <code>0.0</code>
    </Error>
  </Reply>
</ResponseMessage>
```

Figure 60 – Example of a reply(OperationSet) message, success case

Or this in the error case, as shown in Figure 61:

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>OperationSet</Noun>
    <Timestamp>2012-12-20T09:30:49Z</Timestamp>
    <Source>MDMS</Source>
    <MessageID>929BFC21-06FA-4D7E-94FE-C58B3C94EFAC</MessageID>
    <CorrelationID>D921A053-80C1-4DB6-960E-2603127B7B92</CorrelationID>
  </Header>
  <Reply>
    <Result>FAILED</Result>
    <Error>
      <code>2.12</code>
      <level>FATAL</level>
      <details>UsagePoint ID already exists</details>
      <ID idType="ServiceDeliveryPointID"
          idAuthority="UtilityXYZ" objectType="UsagePoint">UP43639</ID>
      <operationId>2</operationId>
    </Error>
    <Error>
      <code>5.9</code>
      <level>FATAL</level>
      <details>
        Transaction aborted to maintain transactional integrity
      </details>
      <ID idType="MeterBadgeNumber"
          idAuthority="UtilityXYZ" objectType="Meter">C34531</ID>
      <operationId>1</operationId>
    </Error>
    <Error>
      <code>5.9</code>
      <level>FATAL</level>
      <details>
        Transaction aborted to maintain transactional integrity
      </details>
      <ID idType="ServiceDeliveryPointID"
          idAuthority="UtilityXYZ" objectType="UsagePoint">UP43639</ID>
      <operationId>3</operationId>
    </Error>
  </Reply>
</ResponseMessage>
```

Figure 61 – Example of a reply(OperationSet) message, failure case

9 Scheduling actions for future execution

9.1 General

This clause covers how a meter read may be arranged for subsequent execution. It then goes on to discuss how this and other scheduled actions may be cancelled.

9.2 Scheduling a meter read

9.2.1 General

IEC 61968-9 allows for two ways in which a meter read may be be scheduled for execution at some future time.

The first way is to make a get(MeterReadings) request in which the `<TimeSchedule>` `<scheduleInterval><start>` element specifies the desired time of execution, as described in 6.3.5.

A second way is to issue a create(MeterReadSchedule) message. This also allows for a meter read schedule to be defined on a periodic basis rather than as a one-off request. The requested periods may be either regular or irregular.

9.2.2 Message exchange pattern

The message exchange pattern depicted in Figure 62 shows a create(MeterReadSchedule) message being sent to the HES. This message contains all relevant information necessary for scheduling a meter read: a list of meters specified as EndDevices, EndDeviceGroups, UsagePoints or UsagePointGroups; a list of ReadingTypes that specify the kind of data to be read; a set of time schedules.

The HES responds immediately with a reply(MeterReadSchedule) message that denotes whether the created(MeterReadSchedule) message has been accepted or not.

When the time comes for the specified meters to be read, the HES initiates this over the AMI network. When it receives the requested data from the meters, the HES then issues one or more created(MeterReadings) messages containing the requested data back to the MDMS.

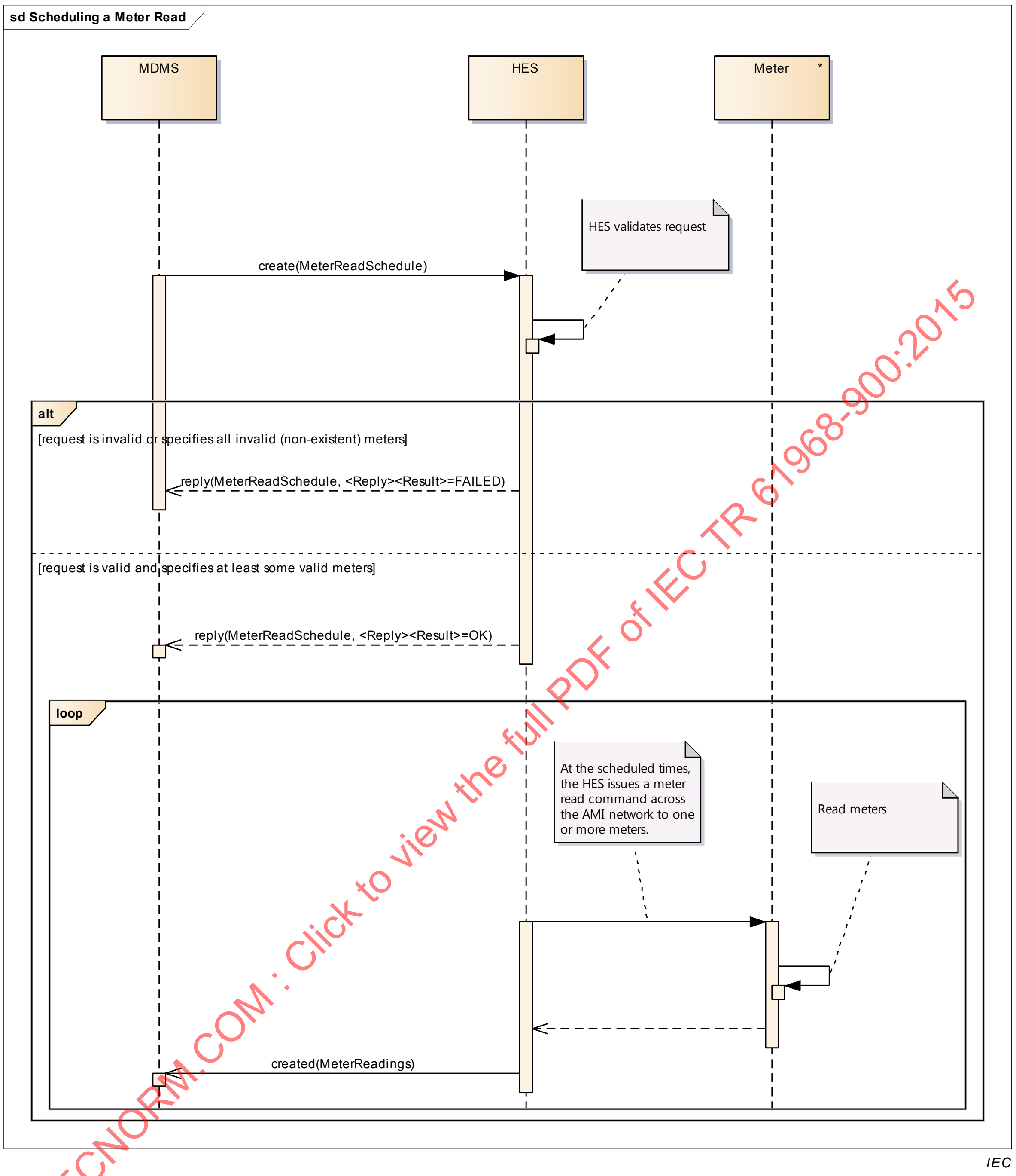

sd Scheduling a Meter Read
MDMS
HES
Meter *
HES validates request
create(MeterReadSchedule)
alt
[request is invalid or specifies all invalid (non-existent) meters]
reply(MeterReadSchedule, <Reply><Result>=FAILED)
[request is valid and specifies at least some valid meters]
reply(MeterReadSchedule, <Reply><Result>=OK)
loop
At the scheduled times, the HES issues a meter read command across the AMI network to one or more meters.
Read meters
created(MeterReadings)
IEC

Figure 62 – Message exchange pattern for scheduling a set of meter reads

9.2.3 MeterReadSchedule element

The overall format of a create(MeterReadSchedule) message is depicted in Figure 63.

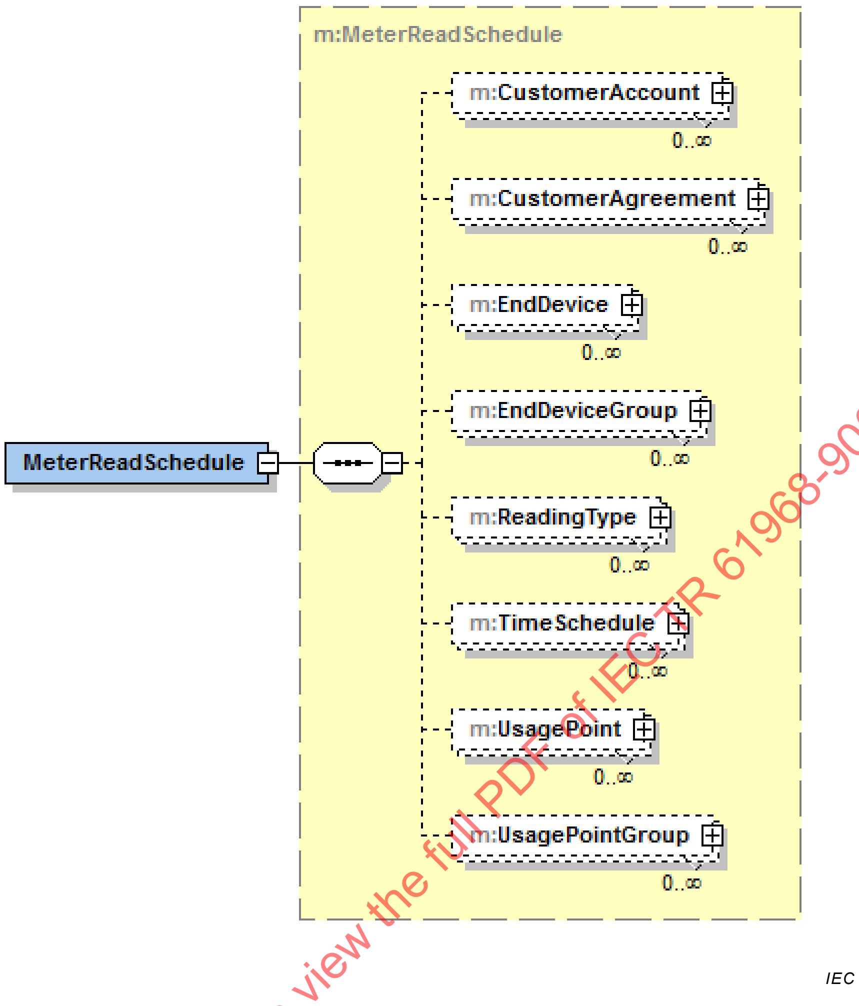
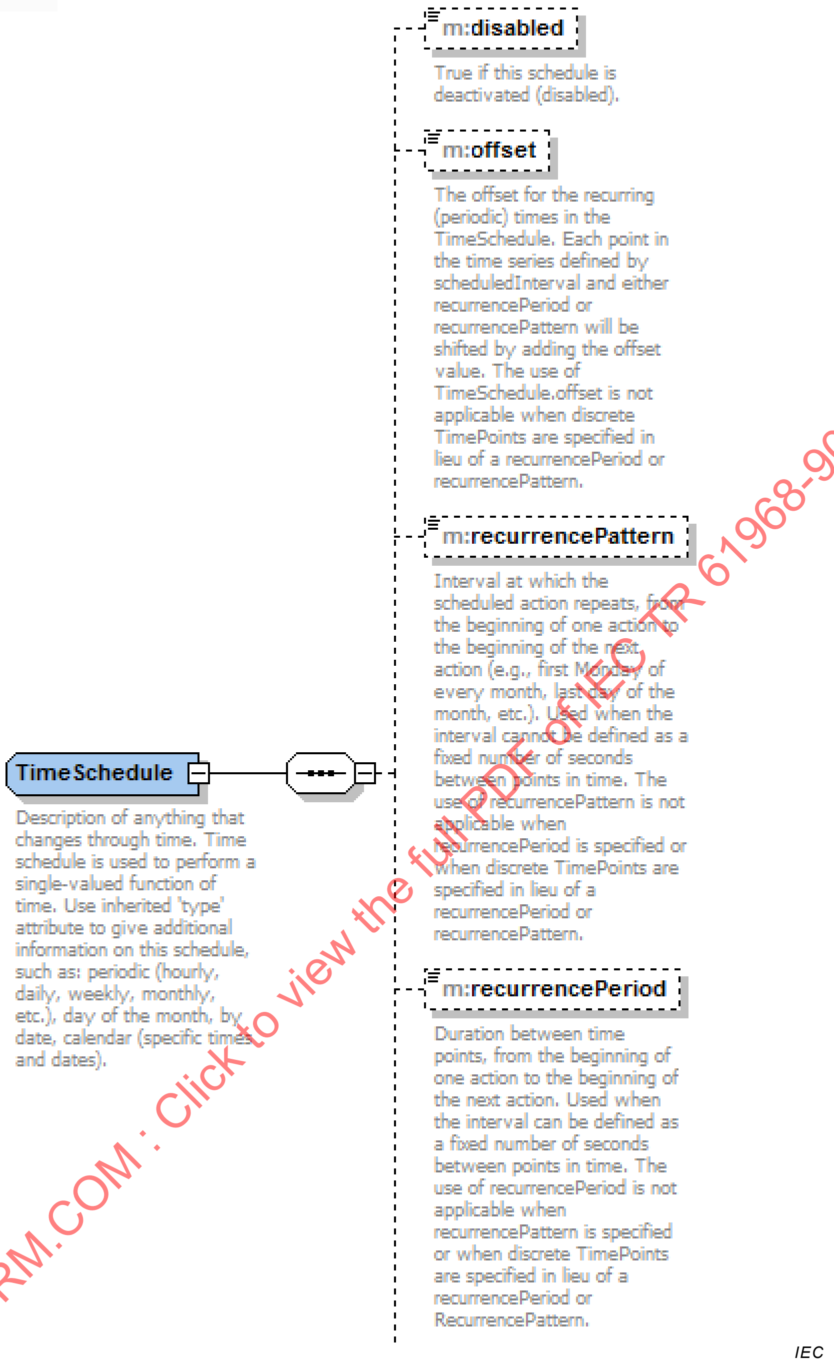


Figure 63 – MeterReadSchedule definition

The format of the TimeSchedule portion of a MeterReadSchedule is shown in Figure 64.


m:disabled
True if this schedule is deactivated (disabled).
m:offset
The offset for the recurring (periodic) times in the TimeSchedule. Each point in the time series defined by scheduledInterval and either recurrencePeriod or recurrencePattern will be shifted by adding the offset value. The use of TimeSchedule.offset is not applicable when discrete TimePoints are specified in lieu of a recurrencePeriod or recurrencePattern.
m:recurrencePattern
Interval at which the scheduled action repeats, from the beginning of one action to the beginning of the next action (e.g., first Monday of every month, last day of the month, etc.). Used when the interval cannot be defined as a fixed number of seconds between points in time. The use of recurrencePattern is not applicable when recurrencePeriod is specified or when discrete TimePoints are specified in lieu of a recurrencePeriod or recurrencePattern.
m:recurrencePeriod
Duration between time points, from the beginning of one action to the beginning of the next action. Used when the interval can be defined as a fixed number of seconds between points in time. The use of recurrencePeriod is not applicable when recurrencePattern is specified or when discrete TimePoints are specified in lieu of a recurrencePeriod or RecurrencePattern.
TimeSchedule
Description of anything that changes through time. Time schedule is used to perform a single-valued function of time. Use inherited 'type' attribute to give additional information on this schedule, such as: periodic (hourly, daily, weekly, monthly, etc.), day of the month, by date, calendar (specific times and dates).
IEC

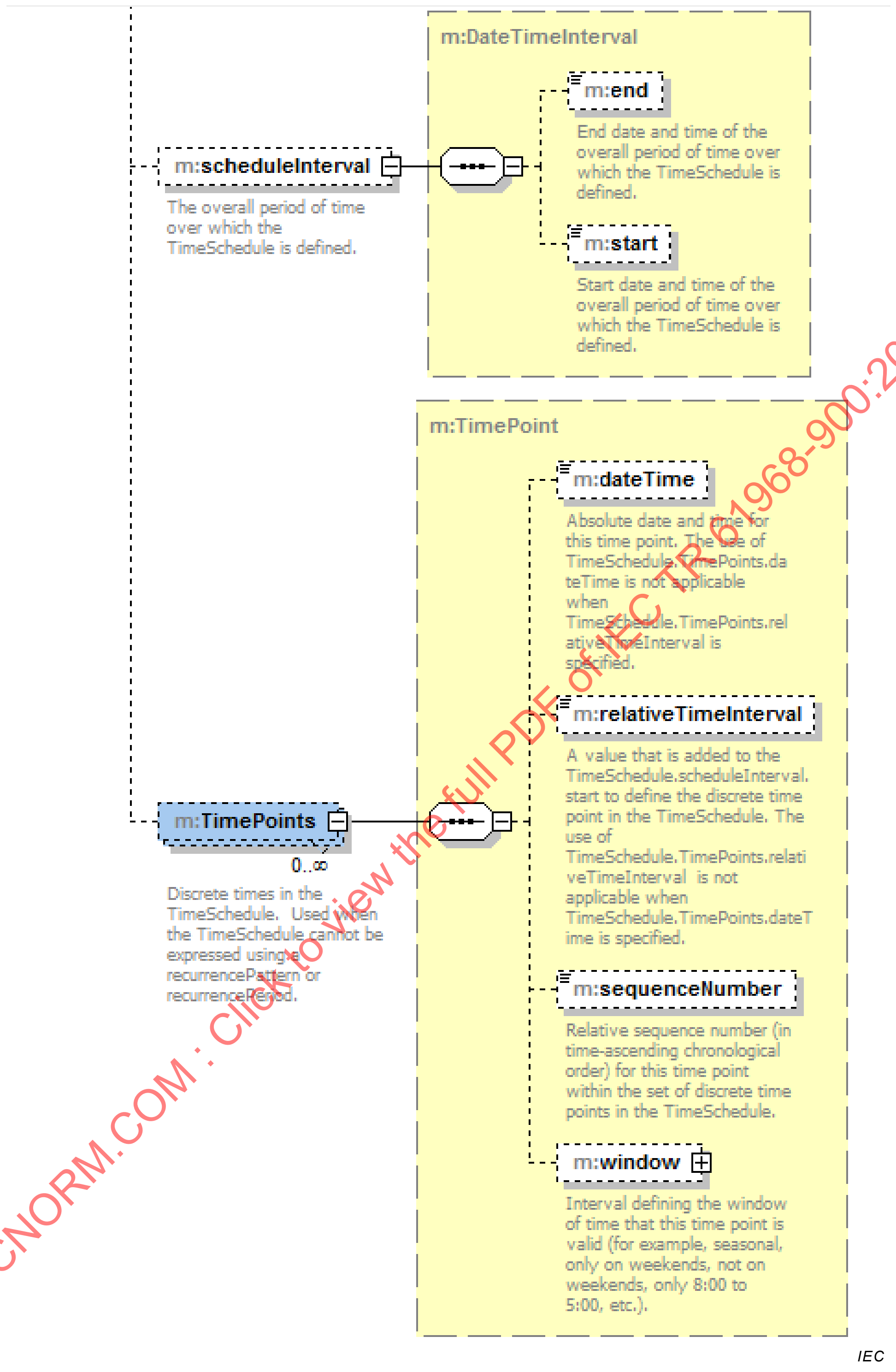

m:DateTimeInterval
m:end
End date and time of the overall period of time over which the TimeSchedule is defined.
m:scheduleInterval
The overall period of time over which the TimeSchedule is defined.
m:start
Start date and time of the overall period of time over which the TimeSchedule is defined.
m:TimePoint
m:dateTime
Absolute date and time for this time point. The use of TimeSchedule.TimePoints.dateTime is not applicable when TimeSchedule.TimePoints.relativeTimeInterval is specified.
m:relativeTimeInterval
A value that is added to the TimeSchedule.scheduleInterval.start to define the discrete time point in the TimeSchedule. The use of TimeSchedule.TimePoints.relativeTimeInterval is not applicable when TimeSchedule.TimePoints.dateTime is specified.
m:TimePoints
0..∞
Discrete times in the TimeSchedule. Used when the TimeSchedule cannot be expressed using a recurrencePattern or recurrencePeriod.
m:sequenceNumber
Relative sequence number (in time-ascending chronological order) for this time point within the set of discrete time points in the TimeSchedule.
m:window
Interval defining the window of time that this time point is valid (for example, seasonal, only on weekends, not on weekends, only 8:00 to 5:00, etc.).
IEC

Figure 64 – MeterReadSchedule.TimeSchedule definition

9.3 Create(MeterReadSchedule) message

9.3.1 General

Figure 65 gives an example of a create(MeterReadSchedule) message. The request is for a specific meter and ReadingType. The schedule request specifies a meter read every day within the given `<scheduleInterval>`. The `<recurrencePeriod>` is measured in seconds.

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>create</Verb>
    <Noun>MeterReadSchedule</Noun>
    <Timestamp>2014-04-03T13:08:15Z</Timestamp>
    <MessageID>2e66f819-703f-44e6-a564-82fe86e7211c</MessageID>
    <CorrelationID>337887b5-f5d3-40d1-a0be-ee3d39bb64f8</CorrelationID>
  </Header>
  <Payload xmlns="http://iec.ch/TC57/2011/MeterReadSchedule#">
    <MeterReadSchedule>
      <EndDevice>
        <mRID/>
        <Names>
          <name>meter1</name>
        </Names>
      </EndDevice>
      <ReadingType>
        <Names>
          <name>0.0.0.1.1.1.12.0.0.0.0.0.0.0.0.3.72.0</name>
        </Names>
      </ReadingType>
      <TimeSchedule>
        <recurrencePeriod>81400</recurrencePeriod>
        <scheduleInterval>
          <end>2014-12-01T00:00:00Z</end>
          <start>2014-12-01T00:00:00Z</start>
        </scheduleInterval>
      </TimeSchedule>
    </MeterReadSchedule>
  </Payload>
</RequestMessage>
```

Figure 65 – Example of a meter read schedule request

9.3.2 Reply(MeterReadSchedule) message

The example given in Figure 66 shows a response message to a previous create(MeterReadSchedule) message. The <Reply><ID> element contains a transaction id whose purpose is explained in 9.4.

```
<ResponseMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>reply</Verb>
    <Noun>MeterReadingSchedule</Noun>
    <Timestamp>2014-04-03T13:08:15Z</Timestamp>
    <MessageID>28e74832-4e90-49c1-9e10-7eb3141b0fcc</MessageID>
    <CorrelationID>337887b5-f5d3-40d1-a0be-ee3d39bb64f8</CorrelationID>
  </Header>
  <Reply>
    <Result>OK</Result>
    <Error>
      <code>0.0</code>
      <level>INFORM</level>
    </Error>
    <ID kind="transaction">6949128301</ID>
  </Reply>
</ResponseMessage>
```

Figure 66 – Example of a response to a meter read schedule request

9.4 Cancelling a request

9.4.1 General

IEC 61968-100 allows for a request to be cancelled before the time for it to start executing has been reached. For example, a scheduled meter read may be cancelled at any time before the specified read time.

The relevant passages in IEC 61968-100:2013 are to be found in Subclauses 6.2.4 and 6.3.3 of that document. These define two main ways in which a cancellation may be made.

First, if a requestor sends a message such as a create(MeterReadSchedule) message to a HES, for example, the latter may reply with a `<Reply><ID>` element. This contains the *transaction id* of the request. The requestor may then cancel the scheduled meter read operation by sending a subsequent cancel(MeterReadSchedule) message that includes the same transaction id in the `<Request><ID>` element.

Alternatively, the requestor may send a create(MeterReadSchedule) message to the HES and itself define the transaction id in the `<Request><ID>` element. This same transaction id may then be used in a subsequent cancel(MeterReadSchedule) message, again in the `<Request><ID>` element. Unfortunately, this puts an imposition on the HES in that it then has to map the transaction id to its own internally generated id and, in the general case, this may not be straightforward to implement.

❗ The current IEC 61968-9 and IEC 61968-100 standards do not make clear which of these possibilities is the normative way of making a cancellation request. Until this is clarified in a subsequent edition of these standards, it is recommend to use the first of the methods presented here.

❗ Cancellations are inherently problematic in that it may be impossible to tell whether a given cancellation request has succeeded or not. If the cancellation is for a request whose action is to take place in the future or which defines an idempotent operation such as a meter read, then there is generally no problem. However, if the cancellation request is for an operation that has already started and has some effect on the meter itself – a create(EndDeviceControls) message, for instance – it may not always be possible to determine which meters has been so affected. Network timing issues can also complicate the picture.

As a general rule, therefore, cancellations are not to be undertaken lightly. The effect of making such a cancellation request is often unpredictable and may leave the overall system in an undefined state.

9.4.2 Cancel(MeterReadSchedule) request

Figure 67 shows an example of a request for cancelling a previous create(MeterReadSchedule) message. The verb is `cancel` and the noun is `MeterReadingSchedule`. The `<Request><ID>` element contains the transaction id that was returned in the previous reply(MeterReadSchedule) message.

```
<RequestMessage xmlns="http://iec.ch/TC57/2011/schema/message">
  <Header>
    <Verb>cancel</Verb>
    <Noun>MeterReadingSchedule</Noun>
    <Timestamp>2014-04-03T13:08:15Z</Timestamp>
    <MessageID>27d4e0d1-9c07-4a4d-a5cc-266e802d7457</MessageID>
    <CorrelationID>337887b5-f5d3-40d1-a0be-ee3d39bb64f8</CorrelationID>
  </Header>
  <Request>
    <ID kind="transaction">6949128301</ID>
  </Request>
</RequestMessage >
```

Figure 67 – Example of a meter read schedule cancel message

10 Transporting IEC 61968-9 messages

10.1 General

IEC 61968-100 mandates how the various messages may be transported from system to system. The current version of this standard deals with web services and JMS transports.

10.2 Transporting over SOAP

10.2.1 General

One of the most common means of transporting messages is to use web services which in turn rely on the SOAP protocol.

For instance, the request message shown in Figure 1 may be embedded in a SOAP message as depicted in Figure 68.

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="http://iec.ch/TC57/2011/schema/message"
    xmlns:gmr="http://iec.ch/TC57/2011/GetMeterReadings#">
  <soapenv:Header/>
  <soapenv:Body>
    <RequestMessage>
      <Header>
        <Verb>get</Verb>
        <Noun>MeterReadings</Noun>
        <Timestamp>2012-10-02T14:16:09Z</Timestamp>
        <MessageID>f1f06eb7-f1a6-463d-b88b-e7474a70631b</MessageID>
        <CorrelationID>10c411ab-b84b-4f13-afd8-f5129f720bc6</CorrelationID>
      </Header>
      <Request>
        <gmr:GetMeterReadings>
          <gmr:EndDevice>
            <gmr:Names>
              <gmr:name>meter1</gmr:name>
            </gmr:Names>
          </gmr:EndDevice>
        </gmr:GetMeterReadings>
      </Request>
    </RequestMessage>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 68 – Example of a simple meter read request imbedded in a SOAP message

The SOAP body contains the IEC 61968-9 message together with the IEC 61968-100 header.

IEC 61968-100 does not mandate what kind of additional information may be conveyed inside the SOAP message header. This might, for example, include authorization information.

10.2.2 Generic WSDL

The "generic WSDL" file forms part of IEC 61968-100. It determines at the outermost level how the various SOAP request and response messages are formulated. As mentioned above, the top-level element of an IEC 61968 message must be one of `<RequestMessage>`, `<ReplyMessage>` or `<EventMessage>`. These elements are mandated by the generic WSDL.

The generic WSDL determines how a web services implementation should respond whenever a request, reply or event message is received. In WSDL parlance all messages are defined as synchronous calls even if the message semantics are asynchronous from an IEC 61968-9 perspective. (This is out of necessity because web services run over HTTP which is a synchronous protocol).

In other words, when a server receives a request message over a web services interface it is expected to respond straightaway with an acknowledgement. This acknowledgement is a reply simply to say "I got your message". A distinct reply containing the requested data or appropriate error notifications will be returned at some later point.

Similarly, when a server sends the requested data or error notifications back to the client system, again over a web services interface, it expects the client to respond straight away with an acknowledgement.

From a messaging perspective, therefore, both the request originator (client) and response originator (server) must be prepared to send and receive SOAP messages.

10.2.3 Simple acknowledgement messages

The form of a simple acknowledgement message is shown in Figure 69. It contains a message header in the `<Header>` element and also a `<Reply>` element. The `<Header><Verb>` is always `reply`. The `<Header><Noun>` and `<Header><CorrelationID>` elements are the same as the corresponding elements in the original request.

The `<Reply>` contains a `<Result>` subelement with value `OK` and an `<Error><code>` subelement with value `0.3`. The latter is defined as meaning "simple acknowledgement".

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="http://iec.ch/TC57/2011/schema/message">
  <soapenv:Header/>
  <soapenv:Body>
    <ResponseMessage>
      <Header>
        <Verb>reply</Verb>
        <Noun>MeterReadings</Noun>
        <Timestamp>2012-10-03T13:08:17Z</Timestamp>
        <MessageID>0a958373-3d47-4f6c-9a5e-34f0da7b94db</MessageID>
        <CorrelationID>10c411ab-b84b-4f13-afd8-f5129f720bc6</CorrelationID>
      </Header>
      <Reply>
        <Result>OK</Result>
        <Error>
          <code>0.3</code>
          <reason>simple ack</reason>
        </Error>
      </Reply>
    </ResponseMessage>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 69 – Example of a simple acknowledgement message

10.2.4 Example message flow

An example of how messages are sent and received according to the generic WSDL specification is shown in Figure 70. This corresponds to the first two messages shown in Figure 17 – that is, a get(MeterReadings) message and its simple acknowledgement reply followed by a reply(MeterReadings) message going in the opposite direction and its simple acknowledgement reply.

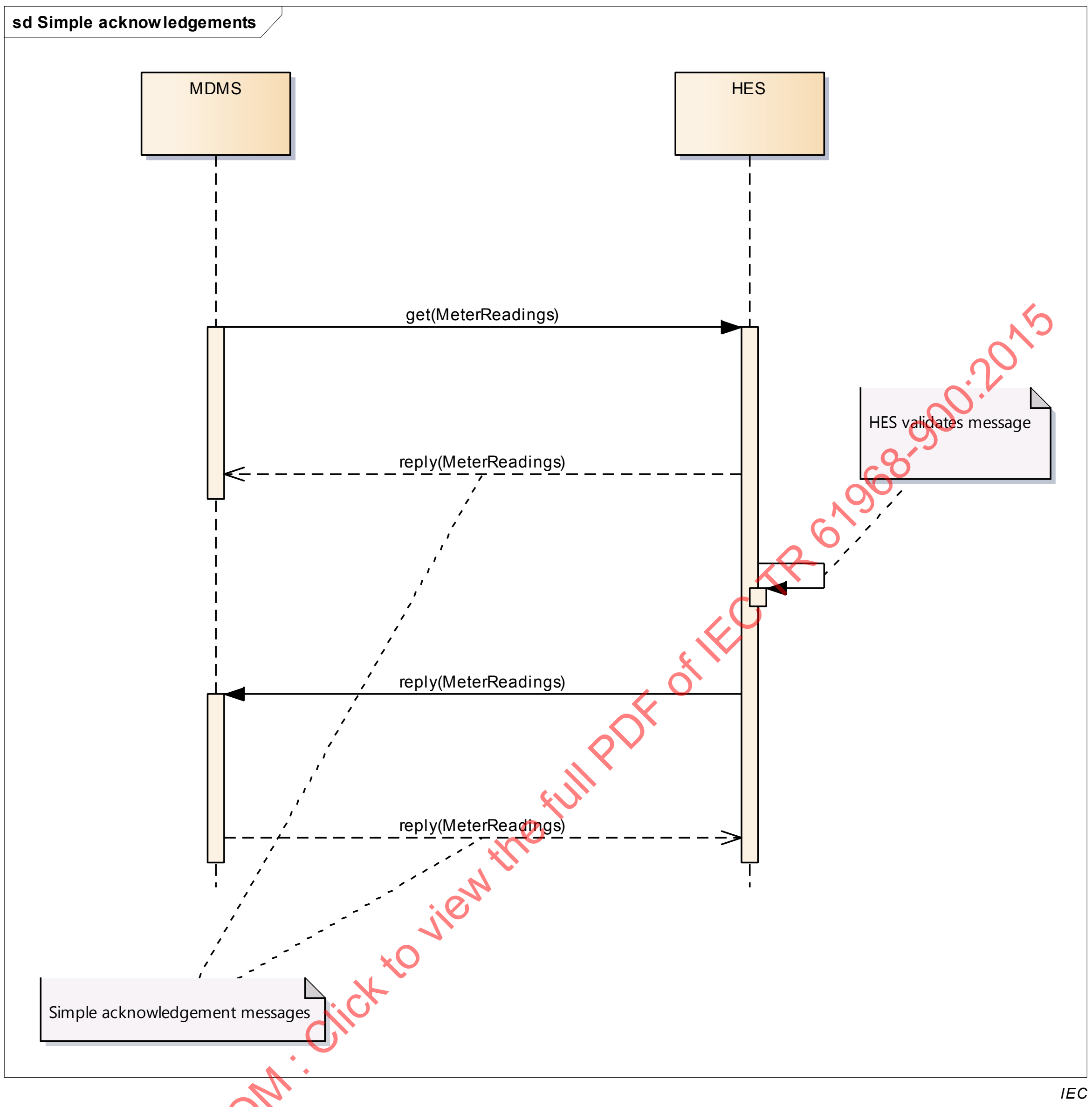


Figure 70 – Message exchange pattern showing the simple acknowledgement messages

Elsewhere in this document the presence or absence of simple acknowledgement messages is ignored. They are clearly important for proper systems integration. However, they do not form part of IEC 61968-9 itself.

10.3 Transporting over JMS

10.3.1 General

IEC 61968-100:2013 explains in detail the usage of JMS as a transport. Of particular relevance are Subclauses 5 and 8.4.

Here an example is provided showing how to configure a JMS to allow an AMI system to publish outage notifications to interested subscribers as described in 7.6.

Subscriptions are made by means of a request to the JMS system for one or more topics. The system administrator must previously have configured and authorized the subscribers for the service being requested. The AMI system then publishes information to the JMS topics and these are picked up by the subscribers.

In the case of outage notifications, these are created(EndDeviceEvent) messages. These notifications are wrapped inside a `javax.jms.ObjectMessage` interface.

10.3.2 Explicit acknowledgements

When using JMS as the transport, generally no explicit acknowledgement message is sent. This is in contrast to using web services with the generic WSDL when a simple acknowledgement message is sent as described in 10.2.2.

When using JMS however, an acknowledgement message can be explicitly requested. The message header should contain an <`AckRequired`> element with text value `true`.

10.3.3 JMS property details

Table 4 lists some example values of the various properties that must be defined when setting up a JMS-based transport.

Table 4 – JMS properties

Property		Value
Topic	Name	AutonomousOutageDetectionTopic
	JNDI Name	jms/topic/AutonomousOutageDetectionTopic
Topic Connection Factory	Name	AutonomousOutageDetectionTopicConnFactory
	JNDI Name	jms/AutonomousOutageDetectionTopicConnFactory
Client ID (for durable subscriber)		<Any Client Id>
Time To Live (TTL)		1 hour
User		<As per server configuration>
Password		<As per server configuration>

A subscriber should use client id while establishing the connection to the topic. The JMS server uses this to identify the subscriber and can send any messages that might have been published on the topic while the subscriber was not running.

The Time To Live (TTL) is the maximum time for which an unconsumed message on the topic will exist inside the JMS system and can be picked up by the subscribers.

10.3.4 Process details

To register with a JMS system and to subscribe to one or more topics, the following course of actions is required:

- Prior to the data consumer (subscriber) presenting a subscription request, the system administrator must configure the data consumer to have subscription privileges for the respective JMS topics.
- The subscriber presents its credentials to the subscription manager via the designated URL.
- The subscriber's credentials are authenticated using the LDAP or a similar mechanism.
- The subscriber's authorizations are confirmed by the AMI system.
- If the authentication and authorization checks out, a subscription is formed and a URL which indicates the place to pick up data is supplied to the data consumer.

On the data producer (publisher) side the following actions take place for each subscription topic:

- Data bubbles up from applications below the JMS interface.

- XML publications are marshaled and placed at the appropriate JMS destinations.
- If data is not picked-up in a reasonable timeframe (as defined by the configuration), the data are deleted and a notification generated. If no subscribers exist for the data, it will not be picked-up so this rule also covers the no-subscribers scenario.
- The nature of JMS topics ensures that the data consumer is not overrun with data.

The publish–subscribe interface supports a timeout constraint so that data which has been placed for pickup by a subscriber can be considered abandoned after a certain time. The interface has a "time to live" parameter (see above) which controls how long data may remain in the topic before being considered abandoned. When a timeout occurs, the abandoned data are deleted and a notification is generated.

10.3.5 Object details

As noted above, the subscription is made using a JMS call which contains the subscriber's credentials. The subscriber calls the appropriate topic factory according to the type of data being sought. The noun and verb normally associated with a subscription are implied in the JMS call.

Data are placed in preconfigured JMS topics. They are available to the subscriber at the destination URL supplied when the subscription was formed. Data are stored in first-in first-out order.

The data are in XML format and conform to an IEC 61968-9 profile appropriate for the flow. The data are not wrapped with SOAP.

11 Summary of message fields

11.1 General

This clause provides detailed listings of the fields that are used in the various IEC 61968-9 messages described in this document.

In this clause the MDMS and HES are used as example systems. This terminology is for illustrative purposes only. Alternative abstract actors could just as easily have been used – for example, the "IEC 61968-9 compliant sending (instigating) system" and the "IEC 61968-9 compliant receiving system".

11.2 Meter read operations

11.2.1 General

See Clause 6 for a description of the message exchange pattern relating to how a meter readings request and the corresponding response are made.

11.2.2 Request message

A request for a meter read is made by sending a message with a header `<Verb>` of `get` and a header `<Noun>` of `MeterReadings`.

The data that specify the qualifying criteria for such a request are passed in the `<Request>` part of the message.

The name of the XSD profile file is `GetMeterReadings.xsd`.

The corresponding namespace is http://iec.ch/TC57/2011/GetMeterReadings#.

Table 5 lists the elements pertaining to a get(MeterReadings) request message.

Table 5 – get(MeterReadings) fields

Element name		XSD data type	Description
Header	Verb	String	`get`
	Noun	String	`MeterReadings`
	Timestamp	dateTime	Date and time this message was created.
	ReplyAddress	String	Address to be used for asynchronous web service replies. This element is omitted when not using web services.
	AckRequired	Boolean	Set to `true` to request an acknowledgement response message from the HES. Set to `false` to have the HES suppress the acknowledgment response message When using web services, this element may be omitted – an acknowledgment response message is always sent.
	MessageID	String	MDMS-specified message identifier
	CorrelationID	String	MDMS-specified identifier used for allowing request-response correlation
Request.GetMeterReadings	EndDevice.Names.name	String	Identifier of the EndDevice (meter) for which readings are being requested. Specify this optional element to limit the result set of readings returned to readings from this EndDevice. Multiple End Devices (Meters) may be specified by repeating the Names class information for each EndDevice.
	EndDevice.Names.NameType.name	String	Identifier of the NameType. Required only if GetMeterReadings.EndDevice.Names.name is not unique in the absence of this additional identifier.
	EndDevice.Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if GetMeterReadings.EndDevice.Names.name coupled with the GetMeterReadings.EndDevice.Names.NameType.name is not unique in the absence of this additional identifier.
	ReadingQualities.ReadingQualityType.Names.name	Enumerated String	3-position ReadingQualityType code as defined by annex D of the IEC 61968-9 specification. Specify this optional element to limit the result set of readings returned to readings having this ReadingQualityType. Multiple ReadingQualityTypes may be specified. If omitted, the request is for all ReadingQualityTypes available in the HES for the specified End Point(s).
	ReadingType.Names.name	Enumerated String	18-position ReadingType code as defined by annex C of the IEC 61968-9 specification . Specify this optional element to limit the result set of readings returned to readings having this ReadingType. Multiple ReadingTypes may be specified by repeating this element. If omitted, the request is for all ReadingTypes available in the HES for the specified End Point(s).

Element name		XSD data type	Description
	TimeSchedule.scheduleInterval.end	dateTime	End date and time of the interval for which readings are being requested. Specify this optional element to limit the result set of readings returned to readings with timestamps in the interval specified by GetMeterReadings.TimeSchedule.scheduleInterval.start and GetMeterReadings.TimeSchedule.scheduleInterval.end. To specify a point in time rather than an interval, set the GetMeterReadings.TimeSchedule.scheduleInterval.start and GetMeterReadings.TimeSchedule.scheduleInterval.end to the same value or omit the GetMeterReadings.TimeSchedule.scheduleInterval.end.
	TimeSchedule.scheduleInterval.start	dateTime	Start date and time of the interval for which readings are being requested. Specify this optional element to limit the result set of readings returned to readings with timestamps in the interval specified by GetMeterReadings.TimeSchedule.scheduleInterval.start and GetMeterReadings.TimeSchedule.scheduleInterval.end. To specify a point in time rather than an interval, set the GetMeterReadings.TimeSchedule.scheduleInterval.start and GetMeterReadings.TimeSchedule.scheduleInterval.end to the same value or omit the GetMeterReadings.TimeSchedule.scheduleInterval.end.
	UsagePoint.Names.name	String	Identifier of a UsagePoint for which readings are being requested. Specify this optional element to limit the result set of readings returned to readings from this UsagePoint. Multiple UsagePoints may be specified by repeating the Names class information for each UsagePoint.
	UsagePoint.Names.NameType.name	String	Identifier of the NameType. Required only if GetMeterReadings.UsagePoint.Names.name is not unique in the absence of this additional identifier.
	UsagePoint.Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if GetMeterReadings.UsagePoint.Names.name coupled with the GetMeterReadings.UsagePoint.Names.NameType.name is not unique in the absence of this additional identifier.

11.2.3 Response message

A response to a meter read request is made by sending a message with a header <Verb> of `reply` and a header <Noun> of `MeterReadings`.

The data for such a response are passed in the <Reply> and <Payload> parts of the message.

The name of the XSD profile file is `MeterReadings.xsd`.

The corresponding namespace is http://iec.ch/TC57/2011/MeterReadings#.

Table 6 lists the elements pertaining to a reply(MeterReadings) response message.

Table 6 – reply(MeterReadings) fields

Element name		XSD data type	Description
Header	Verb	String	`reply`
	Noun	String	`MeterReadings`
	Timestamp	dateTime	Date and time this message was created.
	ReplyAddress	String	Address to be used for asynchronous web service replies. This element is omitted when not using web services.
	AckRequired	Boolean	Set to `true` to request an acknowledgement response message from the HES. Set to `false` to have the HES suppress the acknowledgment response message When using web services, this element may be omitted – an acknowledgment response message is always sent.
	MessageID	String	MDMS-specified message identifier
	CorrelationID	String	MDMS-specified identifier used for allowing request-response correlation
Reply	Result	Enumerated String	See clause 4.3 for a description for how these fields should be set.
	Error.code	String	
	Error.level	Enumerated String	
	Error.details	String	The Reply.Error.details for each instance of the Reply.Error structure may optionally be populated with free-form details concerning the fatal error or informational condition.
	Error.ID	String	The Names.name identifier of the EndDevice or UsagePoint for which the fatal error or informational condition is being reported. Required unless the Reply.Error.code is `0.0`, `0.1` or `0.2`.
	Error.ID@idType	String	The Names.NameType.name identifier of the EndDevice (Meter) or UsagePoint for which the fatal error or informational condition is being reported. Required only if the Error.ID is provided and is not unique in the absence of this additional identifier.
	Error.ID@idAuthority	String	The Names.NameType.NameTypeAuthority.name identifier of the EndDevice (Meter) or Usage Point for which the fatal error or informational condition is being reported. Required only if Error.ID coupled with Error.ID@idType is provided and is not unique in the absence of this additional identifier.
	Error.ID@kind	Enumerated String	Set to `name`. Indicates that the EndDevice (Meter) or UsagePoint is being identified by an instance of the Names class. Required only if Error.ID is provided.
	Error.ID@objectType	String	Set to `Meter` or `UsagePoint` as appropriate. Required only if Error.ID is provided.

Element name		XSD data type	Description
Payload.MeterReadings.MeterReading	IntervalBlocks.IntervalReadings.reportedDateTime	dateTime	Date and time at which the Interval Reading was first delivered to the Metering System. Typically used only when there are detailed auditing requirements.
	IntervalBlocks.IntervalReadings.source	String	System or entity that originally supplied the MeterReadings.MeterReading.IntervalBlocks.IntervalReadings (e.g., customer, HES, handheld reading system, another enterprise system, etc.).
	IntervalBlocks.IntervalReadings.timeStamp	dateTime	The date and time of the end of the interval for this Interval Reading. Required if IntervalReadings is included in the message.
	IntervalBlocks.IntervalReadings.value	String	The value for this Interval Reading. The value is normally supplied, but may be omitted under certain circumstances (e.g., when the MeterReadings.MeterReading.IntervalBlocks.IntervalReadings. ReadingQualityType indicates that the quality is Indeterminate or there is a known missing read.)
	IntervalBlocks.IntervalReadings.ReadingQualities.comment	String	Optional comment explaining why the particular MeterReadings.MeterReading.IntervalBlocks.IntervalReadings. ReadiingQualityType code was assigned.
	IntervalBlocks.IntervalReadings.ReadingQualities.source	String	The HES that applied the particular MeterReadings.MeterReading.IntervalBlocks.IntervalReadings. ReadiingQualityType code.
	IntervalBlocks.IntervalReadings.ReadingQualities.timeStamp	timeStamp	The date and time that the specified MeterReadings.MeterReading.IntervalBlocks.IntervalReadings. ReadiingQualityType code was applied to the MeterReadings.MeterReading.IntervalBlocks.IntervalReadings.
	IntervalBlocks.IntervalReadings.ReadingQualities. ReadingQualityType@ref	Enumerated String	3-position ReadingQualityType code from the first column in annex D that specifies a ReadingQualityType that has been applied to the IntervalReading. Required if ReadingQualities is included in the message.
	IntervalBlocks.IntervalReadings.timePeriod.end	dateTime	Optional element identifying the actual date and time of the end of the period over which a minimum, maximum, average, total or other function of the Interval Reading was determined. Used only when the macroPeriod (1st position of an 18-position ReadingType code from annex C) has a non-zero value indicating a period of time such as “daily”, “weekly”, “monthly”, “billing period”, “seasonal” or “specifiedPeriod”.
	IntervalBlocks.IntervalReadings.timePeriod.start	dateTime	Optional element identifying the actual date and time of the start of the period over which a minimum, maximum, average, total or other function of the Interval Reading was determined. Used only when the macroPeriod (1st position of an 18-position ReadingType code from annex C) has a non-zero value indicating a period of time such as “daily”, “weekly”, “monthly”, “billing period”, “seasonal” or “specifiedPeriod”.
	IntervalBlocks.ReadingType@ref	Enumerated String	The 18-digit ReadingType as defined by annex C for the MeterReadings.MeterReading.IntervalBlocks.IntervalReadings. Required if IntervalBlocks is included in the message.
	Meter.Names.name	String	Unique identifier of the Meter for which the MeterReading is being reported. This field is required to be populated if the original request used the EndDevice (Meter) addressing mode.
	Meter.Names.NameType. name	String	Identifier of the NameType. Required only if the MeterReadings.MeterReading.Meter.Names.name is provided and is not unique in the absence of this additional identifier.

Element name		XSD data type	Description
Payload.MeterReadings.MeterReading	Meter.Names.NameType. NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if the MeterReadings.MeterReading.Meter.Names.name coupled with the MeterReadings.MeterReading.Meter.Names.NameType.name is provided and is not unique in the absence of this additional identifier.
	Readings.reason	Enumerated String	Reason for this MeterReadings.MeterReading.Readings being taken.
	Readings.reportedDate Time	dateTime	Date and time at which the MeterReadings.MeterReading.Readings was first delivered to the Metering System. Typically used for audit trail purposes.
	Readings.source	String	System or entity that originally supplied the MeterReadings.MeterReading.Readings (e.g., customer, HES, handheld reading system, another enterprise system, etc.).
	Readings.timeStamp	dateTime	The date and time when the Meter reading value actually occurred. Required if Readings is included in the message.
	Readings.value	String	The value for this Reading. The value is normally supplied, but may be omitted under certain circumstances (e.g., when the MeterReadings.MeterReading.Readings.ReadingQualityType indicates that the quality is Indeterminate or there is a known missing read.)
	Readings.ReadingQualities. comment	String	Optional comment explaining why the particular MeterReadings.MeterReading.Readings.ReadiingQualityType code was assigned.
	Readings.ReadingQualities. source	String	The HES that applied the particular MeterReadings.MeterReading.Readings.ReadiingQualityType code.
	Readings.ReadingQualities. timeStamp	datetime	The date and time that the specified MeterReadings.MeterReading.Readings.ReadiingQualityType code was applied to the Reading.
	Readings.ReadingQualities. ReadingQualityType@ref	Enumerated String	3-position ReadingQualityType code as defined by annex D that specifies a ReadingQualityType that has been applied to the Reading. Required if ReadingQualities is included in the message.
	Readings.ReadingType@ ref	Enumerated String	18-position ReadingType code as defined by annex C specifying the type of the Reading. Required if Readings is included in the message.
	Readings.timePeriod.end	dateTime	Optional element identifying the actual date and time of the end of the period over which a minimum, maximum, average, total or other function of a MeterReadings.MeterReading.Readings is determined. Used only when the macroPeriod (1st position of an 18-position ReadingType code from annex C) has a non-zero value indicating a period of time such as “daily”, “weekly”, “monthly”, “billing period”, “seasonal” or “specifiedPeriod”.
	Readings.timePeriod.start	dateTime	Optional element identifying the actual date and time of the start of the period over which a minimum, maximum, average, total or other function of a MeterReadings.MeterReading.Readings is determined. Used only when the macroPeriod (1st position of an 18-position ReadingType code from Annex C) has a non-zero value indicating a period of time such as “daily”, “weekly”, “monthly”, “billing period”, “seasonal” or “specifiedPeriod”.

Element name		XSD data type	Description
	UsagePoint.Names.name	String	Unique identifier of the UsagePoint for which the MeterReading is being reported.
	UsagePoint.Names.NameType.name	String	Identifier of the NameType. Required only if the MeterReadings.MeterReading.UsagePoint.Names.name is provided and is not unique in the absence of this additional identifier.
	UsagePoint.Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if the MeterReadings.MeterReading.UsagePoint.Names.name coupled with the MeterReadings.MeterReading.UsagePoint.Names.NameType.name is provided and is not unique in the absence of this additional identifier.

11.2.4 Unsolicited meter read

An unsolicited meter read has the same fields as described for the response message case in 11.2.3 except for two differences:

- The header `<Verb>` is `created`.
- The message contains a `<Header>` and a `<Payload>` but no `<Reply>`.

11.3 Meter control operations

11.3.1 General

See Clause 7 for a description of the message exchange pattern relating to how a meter control request and the corresponding responses are made.

11.3.2 Request message elements

A request for a meter control operation is made by sending a message with a header `<Verb>` of `create` and a header `<Noun>` of `EndDeviceControls`.

The name of the XSD profile file is `EndDeviceControls.xsd`.

The corresponding namespace is `http://iec.ch/TC57/2011/EndDeviceControls#`.

Table 7 lists the elements pertaining to a create(EndDeviceControls) request message.

Table 7 – create(EndDeviceControls) fields

Element name		XSD data type	Description
Header	Verb	String	`create`
	Noun	String	`EndDeviceControls`
	Timestamp	dateTime	Date and time this message was created.
	ReplyAddress	String	Address to be used for asynchronous web service replies. This element is omitted when not using web services.
	AckRequired	Boolean	Set to `true` to request an acknowledgement response message from the HES. Set to `false` to have the HES suppress the acknowledgment response message When using web services, this element may be omitted – an acknowledgment response message is always sent.
	MessageID	String	MDMS-specified message identifier
	CorrelationID	String	MDMS-specified identifier used for allowing request-response correlation
Payload.EndDeviceControls.EndDeviceControl	issuerID	String	Unique identifier of the business entity originating the EndDeviceControl.
	issuerTrackingID	String	Identifier assigned by the initiator (e.g. retail electric provider) of an end device control action to uniquely identify the demand response event, text message, or other subject of the control action. Can be used when cancelling an event or text message request or to identify the originating event or text message in a consequential end device event.
	reason	String	Reason for the EndDeviceControl request. For a demand reset request, this element may be provided for informational and/or audit trail purposes, but it does not affect the processing of the control request.
	EndDeviceControlType@ref	Enumerated String	4-position EndDeviceControlType code from annex C of the IEC 61968-9 standard that specifies the specific EndDeviceControl action to be performed.
	EndDevices.Names.name	String	Identifier of the Meter for which the demand reset is being requested. Multiple Meters may be specified by repeating the Names class information for each Meter.
	EndDevices.Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if EndDeviceControls.EndDeviceControl.EndDevices.Names.name coupled with the EndDeviceControls.EndDeviceControl.EndDevices.Names.NameType.name is not unique in the absence of this additional identifier.
	primaryDeviceTiming.interval.start	dateTime	Omit this element if this EndDeviceControl action is to be performed immediately. To schedule this EndDeviceControl action for execution at a future date / or time, populate this element with the date and time at which the control action is to be executed.
	UsagePoints.Names.name	String	Identifier of the UsagePoint having the Meter for which the demand reset is being requested. Multiple UsagePoints may be specified by repeating the Names class information for each UsagePoint.
	UsagePoints.Names.NameType.name	String	Identifier of the NameType. Required only if EndDeviceControls.EndDeviceControl.UsagePoints.Names.name is not unique in the absence of this additional identifier.

	Element name	XSD data type	Description
	UsagePoints.Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if EndDeviceControls.EndDeviceControl.UsagePoints.Names.name coupled with the EndDeviceControls.EndDeviceControl.UsagePoints.Names.NameType.name is not unique in the absence of this additional identifier.

11.3.3 Initial response message

The first response to a meter control request is made by sending a message with a header <Verb> of `reply` and a header <Noun> of `EndDeviceControls`.

The name of the XSD profile file is `EndDeviceControls.xsd`.

The corresponding namespace is `http://iec.ch/TC57/2011/EndDeviceControls#`.

Table 8 lists the elements pertaining to a reply(EndDeviceControls) response message.

Table 8 – reply(EndDeviceControls) fields

	Element name	XSD data type	Description
Header	Verb	String	`reply`
	Noun	String	`EndDeviceControls`
	Timestamp	dateTime	Date and time this message was created.
	ReplyAddress	String	Address to be used for asynchronous web service replies. This element is omitted when not using web services.
	AckRequired	Boolean	Set to `true` to request an acknowledgement response message from the HES. Set to `false` to have the HES suppress the acknowledgment response message When using web services, this element may be omitted – an acknowledgment response message is always sent.
	MessageID	String	MDMS-specified message identifier
	CorrelationID	String	MDMS-specified identifier used for allowing request-response correlation
Reply	Result	EnumeratedString	See clause 4.3 for a description for how these fields should be set.
	Error.code	String	
	Error.level	Enumerated String	
	Error.details	String	The Reply.Error.details for each instance of the Reply.Error structure may optionally be populated with free-form details concerning the fatal error or informational condition.
	Error.ID	String	The Names.name identifier of the EndDevice or UsagePoint for which the fatal error or informational condition is being reported. Required unless the Reply.Error.code is `0.0`, `0.1` or `0.2`.
	Error.ID@idType	String	The Names.NameType.name identifier of the EndDevice (Meter) or UsagePoint for which the fatal error or informational condition is being reported. Required only if the Error.ID is provided and is not unique in the absence of this additional identifier.

Element name		XSD data type	Description
	Error.ID@idAuthority	String	The Names.NameType.NameTypeAuthority.name identifier of the EndDevice (Meter) or Usage Point for which the fatal error or informational condition is being reported. Required only if Error.ID coupled with Error.ID@idType is provided and is not unique in the absence of this additional identifier.
	Error.ID@kind	Enumerated String	Set to `name`. Indicates that the EndDevice (Meter) or UsagePoint is being identified by an instance of the Names class. Required only if Error.ID is provided.
	Error.ID@objectType	String	Set to `Meter` or `UsagePoint` as appropriate. Required only if Error.ID is provided.

11.3.4 Subsequent consequential event messages

The second and subsequent response messages to a meter control request are made by sending a message with a header <Verb> of *created* and a header <Noun> of `EndDeviceEvents`.

The name of the XSD profile file is `EndDeviceEvents.xsd`.

The corresponding namespace is `http://iec.ch/TC57/2011/EndDeviceEvents#`.

Table 9 lists the elements pertaining to a created(EndDeviceEvents) event message.

Table 9 – created(EndDeviceEvents) fields

Element name		XSD data type	Description
Header	Verb	String	`created`
	Noun	String	`EndDeviceEvents`
	Timestamp	dateTime	Date and time this message was created.
	ReplyAddress	String	Address to be used for asynchronous web service replies. This element is omitted when not using web services.
	AckRequired	Boolean	Set to `true` to request an acknowledgement response message from the HES. Set to `false` to have the HES suppress the acknowledgment response message When using web services, this element may be omitted – an acknowledgment response message is always sent.
	MessageID	String	MDMS-specified message identifier
	CorrelationID	String	MDMS-specified identifier used for allowing request-response correlation
Payload.EndDeviceEvents.EndDeviceEvent	createdDateTime	dateTime	The date and time that the EndDeviceEvent occurred.
	issuerID	String	This optional element can be populated for tracking / audit trail purposes with the issuerID from the EndDeviceControl in the original request message.
	issuerTrackingID	String	This optional element can be populated for tracking / audit trail purposes with the issuerTrackingID from the EndDeviceControl in te original request message.
	reason	String	AMI-specified reason for this EndDeviceEvent occurring.
	severity	String	AMI-specified severity level of the EndDeviceEvent.

Element name		XSD data type	Description
	Assets.Names.name	String	Unique identifier of the Meter for which the event has been detected. Either the EndDeviceEvents.EndDeviceEvent.UsagePoint.Names.name (and potentially the associated NameType.name and NameType.NameTypeAuthority.name) or the EndDeviceEvents.EndDeviceEvent.Assets.Names.name (and potentially the associated NameType.name and NameType.NameTypeAuthority.name) may be provided.
	Assets.Names.NameType.name	String	Identifier of the NameType. Required only if the EndDeviceEvents.EndDeviceEvent.Assets.Names.name is provided and is not unique in the absence of this additional identifier.
	Assets.Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if the EndDeviceEvents.EndDeviceEvent.Assets.Names.name coupled with the EndDeviceEvents.EndDeviceEvent.Assets.Names.NameType.name is provided and is not unique in the absence of this additional identifier.
	EndDeviceEventDetails.name	String	One or more EndDeviceEventDetails name-value pairs may be included to provide supplemental information concerning the event. Care should be exercised in utilizing this functionality as the sending and receiving systems must have a common understanding of the allowable content of the name and value strings.
	EndDeviceEventDetails.value	String	One or more EndDeviceEventDetails name-value pairs may be included to provide supplemental information concerning the event. Care should be exercised in utilizing this functionality as the sending and receiving systems must have a common understanding of the allowable content of the name and value strings.
	EndDeviceEventType@ref	Enumerated String	Unique EndDeviceEventType from annex D of the IEC 61968-9 standard.
	Names.name	String	AMI-assigned unique identifier of the EndDeviceEvent.
	Names.NameType.name	String	Identifier of the NameType. Required only if the EndDeviceEvents.EndDeviceEvent.Names.name is provided and is not unique in the absence of this additional identifier.
	Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if the EndDeviceEvents.EndDeviceEvent.Names.name coupled with the EndDeviceEvents.EndDeviceEvent.Names.NameType.name is provided and is not unique in the absence of this additional identifier.
	UsagePoint.Names.name	String	Unique identifier of the UsagePoint for which the event has been detected. Either the EndDeviceEvents.EndDeviceEvent.UsagePoint.Names.name (and potentially the associated NameType.name and NameType.NameTypeAuthority.name) or the EndDeviceEvents.EndDeviceEvent.Assets.Names.name (and potentially the associated NameType.name and NameType.NameTypeAuthority.name) must be provided. It is acceptable to provide both.
	UsagePoint.Names.NameType.name	String	Identifier of the NameType. Required only if the EndDeviceEvents.EndDeviceEvent.UsagePoint.Names.name is provided and is not unique in the absence of this additional identifier.

Element name		XSD data type	Description
	UsagePoint.Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if the EndDeviceEvents.EndDeviceEvent.UsagePoint.Names.name coupled with the EndDeviceEvents.EndDeviceEvent.UsagePoint.Names.NameType.name is provided and is not unique in the absence of this additional identifier.

11.3.5 Unsolicited meter event

Apart from the absence of the `<CorrelationID>` element in the message header, an unsolicited meter event has identical fields as described for the response message case in 11.3.4.

11.4 Configuration and provisioning

11.4.1 General

See Clause 8 for a description of the messages that may be sent for configuring and provisioning a HES.

This clause shows the addition of new configuration information into a system. The header `<Verb>` in all cases is `create`. However, existing configuration may be altered or removed altogether using `change` or `delete` as the respective verbs.

11.4.2 Provisioning a meter

A request to provision a meter is made by sending a message with a header `<Verb>` of `create` and a header `<Noun>` of `MeterConfig`.

The name of the XSD profile file is `MeterConfig.xsd`.

The corresponding namespace is `http://iec.ch/TC57/2011/MeterConfig#`.

Table 10 lists the elements pertaining to a create(MeterConfig) request message.

Table 10 – create(MeterConfig) fields

Element name		XSD data type	Description
Header	Verb	String	`create`
	Noun	String	`MeterConfig`
	Timestamp	dateTime	Date and time this message was created.
	ReplyAddress	String	Address to be used for asynchronous web service replies. This element is omitted when not using web services.
	AckRequired	Boolean	Set to `true` to request an acknowledgement response message from the HES. Set to `false` to have the HES suppress the acknowledgment response message When using web services, this element may be omitted – an acknowledgment response message is always sent.
	MessageID	String	MDMS-specified message identifier
	CorrelationID	String	MDMS-specified identifier used for allowing request-response correlation

Element name		XSD data type	Description
Payload.MeterConfig	ComFunction.direction	Enumerated String	Set to one of the enumerated values, e.g., `biDirectional`
	ComFunction.enabled	Boolean	Set to true if the meter's communications functions are enabled.
	ComFunction.technology	Enumerated String	Set to one of the enumerated values, e.g., `rfMesh`
	ComFunction.Names.name	String	Set to a value that is unique within this message so that it can be referenced in the MeterConfig.Meter.ComFunction@ref element below. Required only if details of the ComFunction are included in this message.
	Meter.amrSystem	String	Enter the name of the system that is providing automated reads for this meter.
	Meter.isVirtual	Boolean	Set to true if the Meter is a Virtual Meter. If omitted, a value of false is assumed.
	Meter.ConfigurationEvents.createdDateTime	DateTime	Date and time the command to create the Meter was generated.
	Meter.ConfigurationEvents.effectiveDateTime	DateTime	Date and time the MDMS shall consider the Meter to have been or to be created. If this date/time is in the future, MDMS will pend the transaction and process it at the specified future date and time.
	Meter.ConfigurationEvents.effectiveEndDateTime	DateTime	Date and time the MDMS shall consider the Meter to no longer be available for use. Supplied only if an end to the interval of time that the Meter is to be considered valid and usable is known at the time of the creation request.
	Meter.ConfigurationEvents.Names.name	String	Identifier of the ConfigurationEvent that can be used for tracking/auditing processes.
	Meter.ConfigurationEvents.Names.NameType.name	String	Identifier of the NameType. Required only if the MeterConfig.Meter.ConfigurationEvents.Names.name is provided and is not unique in the absence of this additional identifier.
	Meter.ConfigurationEvents.Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if MeterConfig.Meter.ConfigurationEvents.Names.name coupled with the MeterConfig.Meter.ConfigurationEvents.Names.NameType.name is provided and is not unique in the absence of this additional identifier.
	Meter.CustomAttributes.name	String	"CustomAttribute1". Subsequent CustomAttributes will have names of CustomAttribute2, CustomAttribute3, etc. Use of CustomAttributes is utility-specific and provided solely for backwards compatibility with legacy data synchronization interfaces.
	Meter.CustomAttributes.sequenceNumber	Integer	"1". Subsequent CustomAttributes will have sequenceNumbers of "2", "3", etc. Required only if a CustomAttribute with CustomAttribute.names of "CustomAttribute1" is provided.
	Meter.CustomAttributes.value	String	Utility specific content for this CustomAttribute. Required only if a CustomAttribute with CustomAttributes.names of "CustomAttribute1" is provided.
	Meter.ComFunction@ref	String	Set the ref equal to the MeterConfig.ComFunction.Names.name specified above. Required only if this message contains details of the ComFunction.
	Meter.SimpleEndDeviceFunction@ref	String	Set the ref equal to the MeterConfig.SimpleEndDeviceFunction.Names.name specified below. Required only if this message contains details of the SimpleEndDeviceFunction.

Element name		XSD data type	Description
	Meter.EndDeviceInfo.ratingClass	Enumerated String	Set to one of the enumerated values, e.g., “200Amp”.
	Meter.EndDeviceInfo.AssetModel.Manufacturer.Names.name	String	Set to the name of the Meter manufacturer.
	Meter.EndDeviceInfo.capability.autonomousDst	Boolean	Set to true if the Meter automatically adjusts for DaylightSavingsTime. If omitted, a value of false is assumed.
	Meter.EndDeviceInfo.capability.electricMetering	Boolean	Set to true if this is an electricity meter. If omitted, a value of false is assumed; therefore, one of following Booleans should be included and set to true: electricMetering, gasMetering, waterMetering.
	Meter.EndDeviceInfo.capability.onRequestRead	Boolean	Set to true if on-demand reads of this meter are supported.
	Meter.lifecycle.manufacturedDate	DateTime	Set to the date of manufacture of the Meter.
	Meter.MeterMultipliers.kind	Enumerated String	Set to one of the enumerated values, e.g., “kH”.
	Meter.MeterMultipliers.value	Float	Set to the value of the Meter Multiplier, in this case the value of the kH constant.
	Meter.Names.name	String	Identifier of the Meter.
	Meter.Names.NameType.name	String	
	Meter.Names.NameType.NameTypeAuthority.name	String	Identifier of the NameTypeAuthority. Required only if MeterConfig.Meter.Names.name coupled with the MeterConfig.Meter.Names.NameType.name is provided and is not unique in the absence of this additional identifier.
	SimpleEndDeviceFunction.kind	Enumerated String	Set to one of the enumerated values, e.g., “electricMetering”.
	Meter.SimpleEndDeviceFunction.Names.name	String	Set to a value that is unique within this message so that it can be referenced in the MeterConfig.Meter.SimpleEndDeviceFunction@ref element above. Required only if details of the SimpleEndDeviceFunction are included in this message.

11.4.3 Creation of a ComModule

A request to configure a communications module is made by sending a message with a header <Verb> of `create` and a header <Noun> of `ComModuleConfig`.

The name of the XSD profile file is `ComModuleConfig.xsd`.

The corresponding namespace is `http://iec.ch/TC57/2011/ComModuleConfig#`.

Table 11 lists the elements pertaining to a create(ComModuleConfig) request message.

Table 11 – create(ComModuleConfig) fields

Element name		XSD data type	Description
Header	Verb	String	`create`
	Noun	String	`ComModuleConfig`
	Timestamp	dateTime	Date and time this message was created.
	ReplyAddress	String	Address to be used for asynchronous web service replies. This element is omitted when not using web services.