Colibri semantic core: Interface

Version 1.2.2

Date: 2016-08-10

Author: Daniel Schachinger

This document contains the description of the interface as part of the Colibri semantic core. The available message types and their structure are listed.

1 Introduction

Colibri is an open source software project that aims at providing an agile and flexible building energy management. Colibri extensively uses semantically enriched information about the building (e.g. structure, physical characteristics), its building automation systems (e.g. sensors, actuators, and controllers), other energy-consuming equipment and devices (e.g. lighting system), and surrounding systems (e.g. smart grid agents, weather service providers).

The main components of Colibri are the optimizer in the form of a building energy management system (BEMS) and the semantic data store for all relevant information. This data store is surrounded by a uniform interface that manages the access to the semantic information. By implementing this interface, many different systems and system components can be connected to Colibri in order to provide the BEMS with additional knowledge that is necessary to optimize building energy usage. Examples are Web service providers for weather data or agents of the smart grid, such as energy retailers publishing energy price information. In addition, building automation systems (BASs) can be linked to Colibri in order to implement the elaborated measures and provide data from the individual devices within the building. An overview on the general architecture is shown in Figure 1.

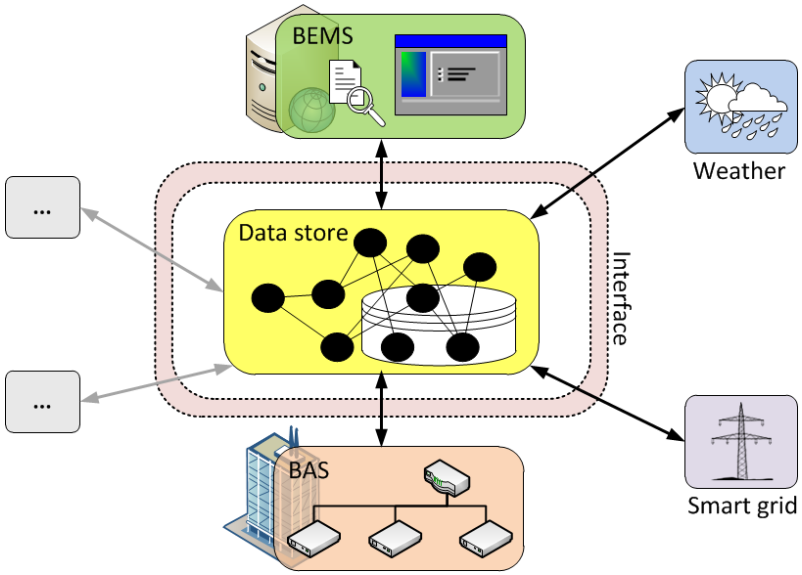

Figure 1: General architecture

The interface in combination with the semantic data store forms the Colibri semantic core. This central component provides all information for further processing and is the data sink for collection of monitoring data from BASs as well as other information from the Web or the smart grid. This document is focused on the description of the interface of the Colibri semantic core in order to provide a specification document for the implementation of technology connectors to bridge the gap between the Colibri world and the domain of external systems and technologies (e.g. BAS, BEMS, or engineering tools).

2 Architecture

In this section, the architecture of the interface is explained in more detail. The interface specification covers the communication between the Colibri system (i.e. the Colibri semantic core) and the technology connectors to other (external) technologies and systems. As can be seen in Figure 2, arbitrary connectors can be linked to the Colibri interface. The message exchange within the connected systems is hidden by the connector. This means that any detailed, specific knowledge of the external technologies is required within the Colibri boundaries. However, the message exchange between the connectors and the Colibri interface needs to be standardized in order to enable uniform connection of all connectors.

As an example, a technology connector to a building's BACnet system, which is responsible for controlling the heating, ventilation, and air conditioning (HVAC) devices, needs to be implemented. For this purpose, a connector component is developed that is able to communicate with BACnet systems. On the other hand, this connector implements the Colibri interface specification. Thus, an instance of this connector is able to establish a link between an actual BACnet system and a running Colibri system. As a consequence, data can be exchanged between these systems using the BACnet connector.

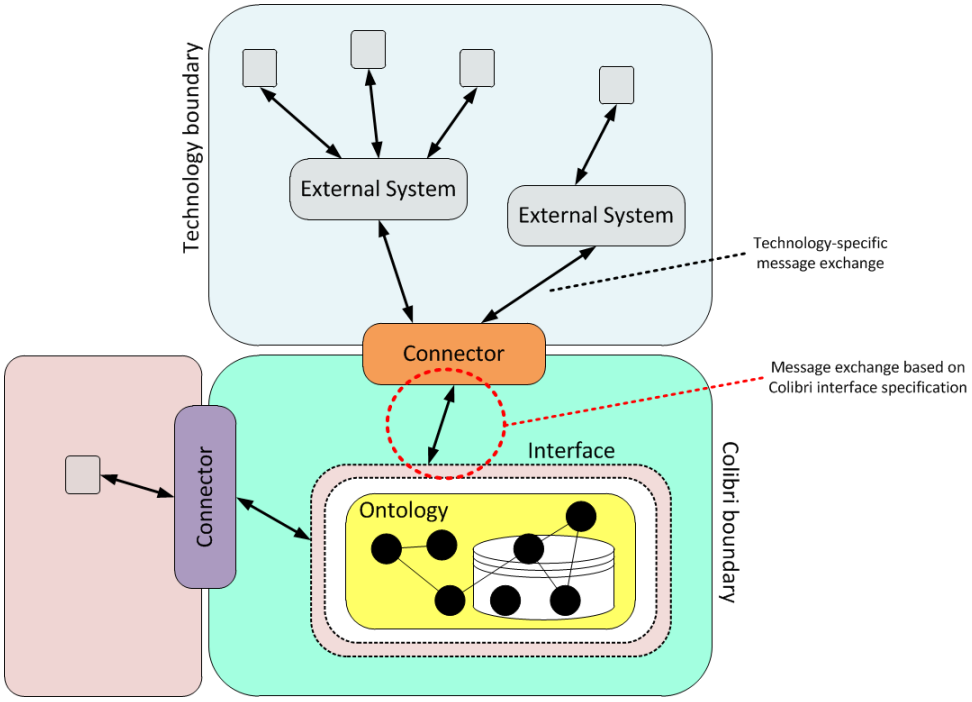

Figure 2: Interface architecture

In order to connect to the Colibri interface, a technology connector has to implement message exchange based on the WebSocket protocol. This allows for a bidirectional communication from the external system to the Colibri system and vice versa without the necessity of polling. Below the WebSocket protocol, the well-known TCP/IP protocol stack is used. Thus, normal IP and Internet infrastructure can be reused for communication purposes. The set of messages, their structure as well as their content are discussed in the following.

3 Message structure

In general, messages are sent from a sender to a receiver. Both the Colibri semantic core and any technology connector can be sender and receiver of messages. In Listing 1, the generic message structure is illustrated. This message structure, which is based on plain text, is chosen in order to keep a certain level of clarity as different encodings can be used in the message content. The structure uses less characters than, for example, a JSON-based or an XML-based message structure.

```
<Message type>
<Header fields>

<Message content>
```

Listing 1: Message structure

Similar to HTTP, each message starts with the *message type* (e.g. REG). In the following lines, a number of case-insensitive *header fields* and their values can be listed. Each header field and its value are written in a distinct line. Lines are separated by "\n". Available header fields are:

- **Message-Id** (mandatory) is used to identify a message. This identifier is unique within the scope of the message sender. Any alphanumeric character (i.e. letters and digits) can be used.
  Example: `Message-Id: 7871ixv.`

- **Content-Type** (mandatory) represents the MIME type of the message content. Depending on the message type, the following content types are supported:
  - text/plain
  - application/x-turtle
  - application/rdf+xml
  - application/sparql-query
  - application/sparql-update
  - application/sparql-result+json
  - application/sparql-result+xml

  An additional charset can be added. The content type header field is mandatory as there is always a message content.
  Example: `Content-Type: application/rdf+xml; charset=utf-8`

- **Date** (optional) shows the date and time that the message was originated using ISO 8601. The used format is *"<date>T<time>Z"* with date format *"YYYY-MM-DD"* and time format *"hh:mm:ss"*. The value is given in UTC.
  Example: `Date: 2016-05-20T11:35:45Z`

- **Expires** (optional) gives the date and time after which the message is run off. The format is the same as for the date header field.
  Example: `Expires: 2016-05-20T11:35:45Z`

- **Reference-Id** (optional/mandatory) is used to specify the identifier of a preceding message the current message is referred to. For example, the reference identifier within an STA message refers to the message that led to this status message. This header field is usually optional. However, if a message is sent in response to another message (e.g. PUT after a GET call, QRE in response to a QUE message), this header field is mandatory.
  Example: `Reference-Id: 7871ixv`

After an empty line (*"\n\n"*), the *message content* is added to the message. This content is encoded in conformance with the specified content type. For resource identifiers in the message content (i.e. address of a targeted resource), the unique resource identifiers (URIs) of the Colibri semantic core are used.

4 Message types

Prior to any message exchange, a secure WebSocket (WSS) connection over Transport Layer Security (TLS) needs to be established by the technology connector. The Colibri semantic core provides the WebSocket endpoint. Once this connection is created, the following messages can be exchanged between the Colibri semantic core (SC) and a technology connector (TC).

Name	**Register connector**
Description	This message is used to give the receiver details about the sender. This information can be used by the receiver to create an instance of the sender in its data store or manage access permissions to the data store. An STA message returns the status of the message reception. Other message types can only be sent after a successful registration.
Message identifier	**REG**
Direction	SC ← TC
Message content	RDF graph containing sender description according to the Colibri ontology
Content type	RDF/XML (application/rdf+xml)[1], Turtle (application/x-turtle)[2]
Response message types	STA

Name	**Deregister connector**
Description	A previously registered connector can be deregistered with this message. The handling of this message is not specified within this document. Both sender and receiver have to run necessary measures when canceling a registration. STA message responses the result of this deregistration.
Message identifier	**DRE**
Direction	SC ⇆ TC
Message content	URI of TC that will be deregistered
Content type	Plain text (text/plain)
Response message types	STA

Name	**Add service**
Description	This message is used to inform the receiver about new services that are available at the sender of the message. This information can be used by the receiver to create instances in its local storage. STA message confirms correct reception or any error that occurred. If a service is initially defined in the Colibri semantic core, an ADD message will be sent to the corresponding technology connector. On the other hand, if the service engineering is done at the technology connector side, the ADD message with the service details is pushed from the technology connector to the Colibri semantic core.
Message identifier	**ADD**
Direction	SC ⇆ TC
Message content	RDF graph containing data service or control service descriptions according to the Colibri ontology
Content type	RDF/XML (application/rdf+xml), Turtle (application/x-turtle)
Response message types	STA

[1] https://www.w3.org/TR/rdf-syntax-grammar/
[2] https://www.w3.org/TR/turtle/

Name	**Remove service**
Description	A previously registered service can be unregistered with this message. For example, the receiver can remove the service from its storage if a service with the given URI is found. Only available services of the particular sender are accepted. STA indicates if any error occurred or reception was successful.
Message identifier	**REM**
Direction	SC ⇆ TC
Message content	URI of data service or control service that will be removed
Content type	Plain text (text/plain)
Response message types	STA

Name	**Observe service**
Description	A previously registered service can be marked for value changes. Then, changes are observed by the receiver of the OBS message. If any change is observed, the sender of the OBS message is informed by receiving an ordinary PUT message. If the service is successfully marked for observation, an STA message with a positive status code is sent. Otherwise an error status code is returned. The sender of the OBS message can optionally define the frequency of sent PUT messages by adding the query parameter *freq* to the message content.
Message identifier	**OBS**
Direction	SC ⇆ TC
Message content	URI of data service or control service that will be observed followed by an optional query parameter (*freq*) defining the frequency of sent PUT messages. If the parameter is not set, PUT messages are automatically sent in case of value changes. If the parameter specifies a time value (XML time datatype) using the format *"hh:mm:ssZ"* (UTC), a PUT message will be sent once a day at the specified point in time. If the parameter defines a duration (XML duration datatype), PUT messages are periodically sent. The timer starts with the reception of the OBS message. All value changes between two PUT messages are buffered and sent with the next PUT message. *Example:* `http://test.org/res1?freq=14:30:00Z`
Content type	Plain text (text/plain)
Response message types	STA

Name	**Detach observation**
Description	This message undoes an observation of the given service. STA message returns the status of the detaching process.
Message identifier	**DET**
Direction	SC ⇆ TC
Message content	URI of already observed data service or control service
Content type	Plain text (text/plain)
Response message types	STA

Name	Put data values
Description	The PUT message is used to exchange simple data values of data as well as control services. These data values conform to a predefined data configuration of services in accordance with the Colibri ontology. For control information, also data values are used as each control service can be a (soft) data service containing the history of state value changes. The service of the actual data value is specified within the message content's RDF graph. STA message with a positive status code is returned if everything worked fine and the PUT message is accepted. STA message with an error status code is sent in case of any error.
Message identifier	**PUT**
Direction	SC ⇆ TC
Message content	RDF graph, which conforms to the Colibri ontology, containing data values for an already added service
Content type	RDF/XML (application/rdf+xml), Turtle (application/x-turtle)
Response message types	STA (optional)

Name	Get data values
Description	A GET message forces the receiver of the message to look for the latest or currently useful value of the given data or control service. For example, the current temperature value is returned, or the currently active set point temperature is sent as response. Moreover, historic data values can be requested by adding the optional query parameters *to* and *from* to the message content. The response is a PUT message with the requested data value(s). STA message can be sent to indicate the status code.
Message identifier	**GET**
Direction	SC ⇆ TC
Message content	URI of already added data or control service followed by optional query parameters (*to*, *from*) specifying date time values in the format *"YYYY-MM-DDThh:mm:ssZ"* (UTC). If none of the parameters of the service's data configuration is of type time, then these query parameters will be ignored. *Example:* `http://test.org/res1?from=2008-01-01T00:00:00Z&to=2010-12-31T23:59:59Z`
Content type	Plain text (text/plain)
Response message types	STA (optional), PUT

Name	Query
Description	This message is used to send a complete SPARQL query (SELECT statement) to the Colibri semantic core. The results are sent back in a QRE message. STA messages can be sent to indicate the status code.
Message identifier	**QUE**
Direction	SC ← TC
Message content	SPARQL query[3]
Content type	SPARQL query (application/sparql-query)[4]
Response message types	STA (optional), QRE

[3] https://www.w3.org/TR/rdf-sparql-query/
[4] https://www.w3.org/TR/sparql11-protocol/

Name	Query result
Description	This message is used to send the result set in response to a QUE message. The receiver can return a status code within an STA message.
Message identifier	**QRE**
Direction	SC → TC
Message content	Result of the previously sent query
Content type	Result set as XML (application/sparql-result+xml)[5] or JSON (application/sparql-result+json)[6]
Response message types	STA (optional)

Name	Update
Description	This message is used to send a complete SPARQL update (INSERT or DELETE statement) to the Colibri semantic core. An STA messages is sent in response in order to indicate the status code.
Message identifier	**UPD**
Direction	SC ← TC
Message content	SPARQL update[7]
Content type	SPARQL update (application/sparql-update) [8]
Response message types	STA

Name	Status
Description	This message sends a status code according to a previously received message.
Message identifier	**STA**
Direction	SC ⇆ TC
Message content	Status code and an optional, human-readable description separated by a blank.
Content type	Plain text (text/plain)
Response message types	-

The Colibri semantic core is able to specify access permissions for each registered technology connector. For example, a KNX connector might not be allowed to send arbitrary UPD messages in order to insert or delete elements in the semantic data store. On the contrary, this connector is allowed to add and remove the own services in the Colibri semantic core. On the other hand, an engineering tool might be allowed to directly edit the content of the Colibri semantic core. Configuration of access permissions is done when a connector has sent its registration message to the Colibri semantic core. The permissions do not have to be limit to message types, but they can also protect certain areas in the semantic data store.

[5] https://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321/
[6] https://www.w3.org/TR/2013/REC-sparql11-results-json-20130321/
[7] https://www.w3.org/Submission/SPARQL-Update/
[8] https://www.w3.org/TR/sparql11-protocol/

5 Status codes

The following status codes can be used within STA messages to inform the receiver of this message about the status of the sender resulting from the reception of a preceding message.

Code	Description
200	OK
300	Errors regarding message structure (e.g. invalid syntax)
400	Syntactical errors regarding message content (e.g. invalid encoding)
500	Semantical errors regarding message content (e.g. contradictions)
600	Connection error (e.g. erroneous transmission)
700	Internal processing errors (e.g. stack overflow)
800	Access permission error (e.g. permissions exceedance)

6 Interaction scenarios

This section shows regular communication scenarios, like the registration of a technology connector or the exchange of new data values. RDF graphs in the following examples are mainly encoded using RDF/XML. The validity of the message content can be checked with the online W3C RDF Validation Service[9].

6.1 Connectors

After establishing the secure WebSocket (WSS) connection, the connector needs to send its characteristics to the Colibri semantic core, where these details are stored in the data store. This is done by sending a register (REG) message to the Colibri semantic core. An example can be seen in Listing 2. Here, a KNX connector is registered. The message content has to conform to the Colibri ontology.

```
REG
Date: 2016-05-15T12:06:49Z
Content-Type: application/rdf+xml
Message-Id: 1605152

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY colibri "https://.../colibri.owl#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:colibri="https://.../colibri.owl#">
  <rdf:Description rdf:about="http://www.colibri-samples.org/tc1">
    <rdf:type rdf:resource="&colibri;KnxConnector"/>
    <colibri:connectorAddress rdf:datatype="&xsd;string">192.168.10.56
    </colibri:connectorAddress>
    <colibri:hasTechnologyProtocol rdf:resource="&colibri;Knx"/>
  </rdf:Description>
</rdf:RDF>
```

Listing 2: REG example

This technology connector description is interpreted by the interface of the Colibri semantic core and a status (STA) message is sent back to the connector. It is assumed that the Colibri semantic core is able to handle this example. Thus, the status code 200 is returned to the connector as can be seen in Listing 3.

```
STA
Date: 2016-05-15T12:06:51Z
Content-Type: text/plain
Message-Id: 1605153
Reference-Id: 1605152

200 OK
```

Listing 3: STA example

In order to deregister a previously registered technology connector, a deregister (DRE) message is sent. The message in Listing 4 is sent from the technology connector, which needs to be deregistered, to the Colibri semantic core. However, deregistration can also be done in the other direction. An STA message is returned in response to the DRE message.

[9] https://www.w3.org/RDF/Validator/

```
DRE
Date: 2016-05-15T17:06:51Z
Content-Type: text/plain
Message-Id: 1605154

http://www.colibri-samples.org/tc1
```

Listing 4: DRE example

6.2 Services

Next to the registration of the technology connector at the Colibri semantic core, both the connector and the Colibri semantic core can exchange information about the connector's services. This is done by sending an add (ADD) message. In Listing 5, an example of an ADD message can be seen that is used to describe an actual data service. Here, the message is sent from the Colibri semantic core to the connector, but also the other direction is possible. The data service description includes the definition of the data configuration. In this example, the data service is able to host data values (i.e. pairs of values) of the parameters temperature and time. The message content has to conform to the Colibri ontology. The unit and other static information used in the data service description are created in the Colibri data store in advance. The receiver sends an STA message in response.

```
ADD
Date: 2016-05-16T14:03:20Z
Content-Type: application/rdf+xml
Message-Id: 1605161

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY colibri "https://.../colibri.owl#">]>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:colibri="https://.../colibri.owl#">
  <rdf:Description rdf:about="http://www.colibri-samples.org/service1">
    <rdf:type rdf:resource="&colibri;BuildingData"/>
    <rdf:type rdf:resource="&colibri;DataService"/>
    <colibri:serviceAddress rdf:datatype="&xsd;string">1.0.4</colibri:serviceAddress>
    <colibri:identifier rdf:datatype="&xsd;string">temp_monitoring_17
    </colibri:identifier>
    <colibri:hasDataConfiguration
             rdf:resource="http://www.colibri-samples.org/configuration1"/>
    <colibri:hasTechnologyConnector rdf:resource="http://www.colibri-samples.org/tc1"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.colibri-samples.org/configuration1">
    <rdf:type rdf:resource="&colibri;DataConfiguration"/>
    <colibri:hasParameter
             rdf:resource="http://www.colibri-samples.org/parameter1"/>
    <colibri:hasParameter
             rdf:resource="http://www.colibri-samples.org/parameter2"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.colibri-samples.org/parameter1">
    <rdf:type rdf:resource="&colibri;TemperatureParameter"/>
    <colibri:hasUnit rdf:resource="&colibri;degree-celsius"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.colibri-samples.org/parameter2">
    <rdf:type rdf:resource="&colibri;TimeParameter"/>
  </rdf:Description>
</rdf:RDF>
```

Listing 5: ADD example

Services that are not maintained anymore can be removed in the technology connector's data store as well as in the Colibri semantic core. For this purpose, the remove (REM) message is utilized. Again, an STA message is sent in response. The REM message can be sent in both directions, and the internal handling of the message depends on the actual implementation and application. An example is shown in Listing 6.

```
REM
Date: 2016-05-17T14:06:51Z
Content-Type: text/plain
Message-Id: 16051727

http://www.colibri-samples.org/service1
```

Listing 6: REM example

Moreover, added services can be observed by the technology connector or the Colibri semantic core. If an observation is added to a service, value or state changes of this service are propagated to the interested communication partner. For example, the Colibri semantic core observes the previously introduced temperature monitoring service. Thus, the technology connector sends new data values to the Colibri semantic core in case of value changes. The observation is initialized with an observe (OBS) message as can be seen in Listing 7.

```
OBS
Date: 2016-05-17T09:03:12Z
Message-Id: 16051721
Content-Type: text/plain

http://www.colibri-samples.org/service1
```

Listing 7: OBS example

Observations are canceled with a detach (DET) message. This message is sent from the initiator of the observation to the receiver of the preceding OBS message. Listing 8 gives an example for this message type. REM, OBS, and DET messages are followed by an STA message back from the receiver to the sender.

```
DET
Date: 2016-05-17T11:23:00Z
Message-Id: 16051722
Content-Type: text/plain

http://www.colibri-samples.org/service1
```

Listing 8: DET example

6.3 Data values

If either a technology connector or the Colibri semantic core wants to get the current or most recent data value of a service, a get (GET) message is sent. For this purpose, the URI of the requested service is sent to the receiver as shown in Listing 9. A history of data values can be requested by adding the optional query parameters *to* or *from* to the services' URI in the message content. The parameter values indicate the start and stop date of the requested history interval. An optional STA message can be returned after receiving and processing a GET message. Moreover, the receiver of the GET message sends a PUT message with the requested data values of the specified service.

```
GET
Date: 2016-05-17T11:25:00Z
Message-Id: 16051723
Content-Type: text/plain

http://www.colibri-samples.org/service1
```

Listing 9: GET example

Such a put (PUT) message contains an RDF graph representing data value(s) according to the Colibri ontology. A PUT message does not need to be preceded by a GET message, but a sender can actively push new values using the PUT message. An optional STA message can be returned by the receiver of a PUT message.

In the example in Listing 10, the PUT message is the response to the previously shown GET message. The text in red can be omitted if the receiver is able to infer the service membership on the basis of the actual values and their connected parameters. If there is only one data value (consisting of two values) transferred by means of a PUT message, also the green text can be omitted. The receiver combines the two values to one data value in its internal representation and adds the data value to the correspondent service. Listing 11 shows the shortened form of this PUT message in Turtle notation.

```
PUT
Date: 2016-05-17T11:25:04Z
Message-Id: 16051724
Reference-Id: 16051723
Content-Type: application/rdf+xml

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY colibri "https://.../colibri.owl#">]>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:colibri="https://.../colibri.owl#">
  <rdf:Description rdf:about="http://www.colibri-samples.org/service1">
    <colibri:hasDataValue rdf:resource="http://www.colibri-samples.org/data-value1"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.colibri-samples.org/data-value1">
    <rdf:type rdf:resource="&colibri;DataValue"/>
    <colibri:hasValue rdf:resource="http://www.colibri-samples.org/value1"/>
    <colibri:hasValue rdf:resource="http://www.colibri-samples.org/value2"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.colibri-samples.org/value1">
    <rdf:type rdf:resource="&colibri;Value"/>
    <colibri:value rdf:datatype="&xsd;float">19.37</colibri:value>
    <colibri:hasParameter rdf:resource="http://www.colibri-samples.org/parameter1"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.colibri-samples.org/value2">
    <rdf:type rdf:resource="&colibri;Value"/>
    <colibri:value rdf:datatype="&xsd;dateTime">2016-05-15T12:37:00Z</colibri:value>
    <colibri:hasParameter rdf:resource="http://www.colibri-samples.org/parameter2"/>
  </rdf:Description>
</rdf:RDF>
```

Listing 10: PUT example

```
PUT
Date: 2016-05-17T11:25:04Z
Message-Id: 16051724
Reference-Id: 16051723
Content-Type: application/x-turtle

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ns1: <http://www.colibri-samples.org/> .
@prefix ns0: <https://.../colibri.owl#> .
ns1:value1
  a ns0:Value ;
  ns0:value "19.37"^^xsd:float ;
  ns0:hasParameter ns1:parameter1 .
ns1:value2
  a ns0:Value ;
  ns0:value "2016-05-15T12:37:00Z"^^xsd:dateTime ;
  ns0:hasParameter ns1:parameter2 .
```

Listing 11: PUT example (shortened)

6.4 Queries and updates

Query (QUE) messages are used to send more advanced queries (SELECT statements) to the Colibri semantic core in order to get detailed information about the semantic information in the data store. The QUE message contains a SPARQL query in the message content. An example is illustrated in Listing 12. SPARQL queries have to be validated[10].

```
QUE
Date: 2016-05-19T15:20:00Z
Message-Id: 16051925
Content-Type: application/sparql-query

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX colibri: <https://.../colibri.owl#>

SELECT ?service ?identifier
WHERE { ?service rdf:type colibri:DataService.
?service colibri:hasDataConfiguration ?y.
?y colibri:hasParameter ?z.
?z rdf:type colibri:TemperatureParameter.
?service colibri:identifier ?identifier}
```

Listing 12: QUE example

The results of queries are returned as query result (QRE) messages. Optionally, the receiver of a QUE message (i.e. the Colibri semantic core) can return an STA message (e.g. in case of an error). Listing 13 shows an example of a QRE message in JSON encoding.

[10] http://www.sparql.org/query-validator.html

```
QRE
Date: 2016-05-19T15:20:10Z
Message-Id: 16051926
Reference-Id: 16051725
Content-Type: application/sparql-result+json

{
 "head": { "vars": [ "service", "identifier" ] },
 "results": {
   "bindings": [
    {
     "service" : { "type": "uri", "value": "http://www.colibri-samples.org/service1" },
     "identifier" : { "type": "literal", "value": "temp_monitoring_17" }
    }
   ]
 }
}
```

Listing 13: QRE example

Finally, the update (UPD) message is used to directly insert or delete triples in the semantic data store. This can be used, for example, by an engineering tool defining the initial building structure. An example of such an UPD message is shown in Listing 14. Validation of SPARQL updates is required[11].

```
UPD
Date: 2016-05-19T15:20:10Z
Message-Id: 16051966
Content-Type: application/sparql-update

PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX colibri: <https://.../colibri.owl#>

INSERT DATA { <http://www.colibri-samples.org/main-building> rdf:type
colibri:OfficeBuilding }
```

Listing 14: UPD example

In summary, this section should give an overview on possible interactions without making any claim to be complete.

[11] http://www.sparql.org/update-validator.html