CoRE
Internet-Draft
Intended status: Standards Track
Expires: January 12, 2013

Z. Shelby
Sensinode
M. Vial
Schneider-Electric

July 11, 2012

CoRE Interfaces
draft-shelby-core-interfaces-03

Abstract

This document defines well-known REST interface descriptions for Batch, Sensor, Parameter and Actuator types for use in contrained web servers using the CoRE Link Format. A short reference is provided for each type that can be efficiently included in the interface description attribute of the CoRE Link Format. These descriptions are intended to be for general use in resource designs or for inclusion in more specific interface profiles. In addition, this document defines the concepts of Function Set and Binding. The former is the basis element to create RESTful profiles and the latter helps the configuration of links between resources located on one or more endpoints.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 12, 2013.

Table of Contents

1. Introduction

The Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). CoRE is aimed at machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC5988]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [I-D.ietf-core-link-format] and can be used by CoAP [I-D.ietf-core-coap] or HTTP servers. The CoRE Link Format defines an attribute that can be used to describe the REST interface of a resource, and may include a link to a description document. This memo describes how other specifications can combine resources with a well-known interface to create new CoRE RESTful profiles. A CoRE profile is based on the concept of Function Set, which is a group of REST resources providing a service in a distributed system. In addition, the notion of Binding is introduced in order to create a synchronization link between two resources. This document also defines well-known interface descriptions for Batch, Sensor, Parameter and Actuator types to compose new Function Sets or for standalone use in a constrained web server. A short reference is provided for each type that can be efficiently included in the interface description (if=) attribute of the CoRE Link Format.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988] and [I-D.ietf-core-link-format]. This specification makes use of the following additional terminology:

Function Set: A group of well-known REST resources that provides a particular service.

Profile: A group of well-known Function Sets defined by a specification.

Device: An IP smart object running a web server that hosts a group of Function Set instances from a profile.

Service Discovery: The process making it possible for a web client to automatically detect devices and Function Sets offered by these devices on a CoRE network.

Resource Discovery: The process allowing a web client to identify resources being hosted on a web server.

Gradual Reveal: A REST design where resources are discovered progressively using Web Linking.

Binding: A unidirectional logical link between a source resource and a destination resource.

3. Function Set

This section defines how a specification can organize REST resources to create a new profile. A profile is structured into groups of resource types called Function Sets. A Function Set is similar to a function block in the sense that it consists of input, output and parameter resources and contains internal logic. A Function Set MAY have a subset of mandatory inputs, outputs and parameters to provide minimum interoperability. It MAY also be extended with manufacturer/ user-specific resources. Other specifications defines the list of function sets available within a given profile. A device is composed of one or more Function Set instances. A profile specification MAY specify device profiles with mandatory function sets.

3.1. Defining a Function Set

In a Function Set, types of resources are defined. Each type includes a human readable name, a path template, a Resource Type for discovery, the Interface Definition and the data type and allowed values. A Function Set definition may also include a field indicating if a sub-resource is mandatory or optional.

3.1.1. Path template

A Function Set is a container resource under which its sub-resources are organized. The profile defines the path to each resource of a Function Set in a path template. The template can contain either relative paths or absolute paths depending on the profile needs. An absolute Function Set SHOULD be located at its recommended root path on a web server, however it MAY be located under an alternative path if necessary (for example multi-purpose devices, gateways etc.). A relative Function Set can be instantiated as many times as needed on a web server with an arbitrary root path. However some Function Sets (e.g. device description) only make sense as singletons.

The path template includes a possible index {#} parameter, and possible fixed path segments. The index {#} allows for multiple instances of this type of resource, and can be any string. The root path and the indexes are the only variable elements in a path template. All other path segments MUST be fixed.

3.1.2. Resource Type

Each root resource of a Function Set is assigned a Resource Type parameter, therefore making it possible to discover it. Each sub-resource of a Function Set is also assigned a Resource Type parameter. This Resource Type is used for resource discovery and is usually necessary to discover optional resources supported on a specific device. The Resource Type of a Function Set may also be used for service discovery and MAY be exported to DNS-SD [I-D.cheshire-dnsext-dns-sd] for example.

The Resource Type parameter defines the value that MUST be included in the rt= field of the CoRE Link Format when describing a link to this resource. The value SHOULD be in the form "namespace.type" for root resources and "namespace.type.subtype" for sub-resources. This naming convention facilitates resource type filtering with the /.well-known/core resource. However a profile MAY allow mixing in foreign namespace references within a Function Set to import external references from other object models (e.g. SenML and UCUM).

3.1.3. Interface Description

The Interface Description parameter defines the REST interface for that type of resource. Several base interfaces are defined in Section 5 of this document. For a given profile, the Interface Description may be inferred from the Resource Type. In that case the Interface Description MAY be elided from link descriptions of resource types defined in the profile, but SHOULD be included for custom extensions to the profile.

The root resource of a Function Set should provide a list of links to its sub-resources in order to offer gradual reveal of resources. The CoRE Link List interface defined in Section 5.1 offers this functionality so a root resource SHOULD support this interface or a derived interface like CoRE Batch (See Section 5.2).

3.1.4. Data type

The Data Type field defines the type of value (and possible range) that is returned in response to a GET for that resource or accepted with a PUT. The interfaces defined in Section 5 make use of plain text and SenML Media types for the actual format of this data. A profile may restrict the list of supported content types for the CoRE interfaces or define new interfaces with new content types.

3.2. Discovery

A device conforming to a profile SHOULD make its resources discoverable by providing links to the resources on the path /.well-known/core as defined in [I-D.ietf-core-link-format]. All resources hosted on a device SHOULD be discoverable either with a direct link in /.well-known/core or by following successive links starting from /.well-known/core.

The root path of a Function Set instance SHOULD be directly referenced in /.well-known/core in order to offer discovery at the first discovery stage. A device with more than 10 individual resources SHOULD only expose Function Set instances in /.well-known/core to limit the size of this resource.

In addition, a device MAY register its resources to a Resource Directory using the registration interface defined in [I-D.shelby-core-resource-directory] if such a directory is available.

3.3. Versioning

A profile should track Function Set changes to avoid incompatibility issues. Evolutions in a Function Set SHOULD be backward compatible.

4. Bindings

In a M2M RESTful environment, endpoints exchange the content of their resources to operate the distributed system. Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human

intervention and a so-called commissioning tool. In this document the abstract relationship between two resources is called a Binding. The configuration phase necessitates the exchange of binding information so a format recognized by all CoRE endpoints is essential. This document defines a format based on the CoRE Link-Format to represent binding information along with the rules to define a binding method which is a specialized relationship between two resources. The purpose of a binding is to synchronize the content between a source resource and a destination resource. The destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address). Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method. The following table gives a summary of the binding methods described in more detail in Section 4.2.

Name	Identifier	Location	Method
Polling	poll	Destination	GET
Observe	obs	Destination	GET + Observe
Push	push	Source	PUT

4.1. Format

Since Binding lies in the creation of a link between two resources, Web Linking and the CoRE Link-Format are a natural way to represent binding information. This involves the creation of a new relation type, purposely named "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource. The Web link attributes allow a fine-grained control of the type of synchronization exchange along with the conditions that trigger an update. This specification defines the attributes below:

Attribute	Parameter	Value
Binding method	bind	xsd:string
Minimum Period (s)	pmin	xsd:integer (>0)
Maximum Period (s)	pmax	xsd:integer (>0)
Change Step	st	xsd:decimal (>0)

Bind Method: This is the identifier of a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

Minimum Period: When present, the minimum period indicates the minimum time to wait (in seconds) before sending a new synchronization message (even if it has changed). In the absence of this parameter, the minimum period is up to the notifier.

Maximum Period: When present, the maximum period indicates the maximum time in seconds between two consecutive syncronization messages (regardless if it has changed). In the absence of this parameter, the maximum period is up to the notifier. The maximum period MUST be greater than the minimum period parameter (if present).

Change Step: When present, the change step indicates how much the value of a resource SHOULD change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification SHOULD be between pmin and pmax.

4.2. Binding methods

A binding method defines the rules to generate the web-transfer exchanges that will effectively send content from the source resource to the destination resource. The description of a binding method must define the following aspects:

Identifier: This is value of the "bind" attribute used to identify the method.

Location: This information indicates whether the binding entry is stored on the source or on the destination endpoint.

REST Method: This is the REST method used in the Request/Response exchanges.

Conditions: A binding method definition must state how the condition attributes of the abstract binding definition are actually used in this specialized binding.

This specification supports 3 binding methods described below.

Polling: The Polling method consists of sending periodic GET requests from the destination endpoint to the source resource and copying the content to the destination resource. The binding entry for this method MUST be stored on the destination endpoint. The destination endpoint MUST ensure that the polling frequency does not exceed the limits defined by the pmin and pmax attributes of the binding entry. The copying process MAY filter out content from the GET requests using value-based conditions (e.g Change Step).

Observe: The Observe method relies on the Publish/Subscribe pattern thus an observation relationship is created between the destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the CoAP Observation mechanism [I-D.ietf-core-observe] hence this method is only permitted when the resources are made available over CoAP. The binding entry for this method MUST be stored on the destination endpoint. The binding conditions are mapped as query string parameters (see Section 5.9).

Push: When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource upon change of the source resource. The source endpoint MUST only send a notification request if the binding conditions are met. The binding entry for this method MUST be stored on the source endpoint.

4.3. Binding table

The binding table is a special resource that gives access to the bindings on a endpoint. A binding table resource MUST support the Binding interface defined in Section 5.8. A profile SHOULD allow only one resource table per endpoint.

5. Interface Descriptions

This section defines REST interfaces for Link List, Batch, Sensor, Parameter, Actuator and Binding table resources. Variants such as Linked Batch or Read-Only Parameter are also presented. Each type is described along with its Interface Description attribute value and valid methods. These are defined for each interface in the table below. These interfaces can support plain text and/or SenML Media types.

The if= column defines the Interface Description (if=) attribute

value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this specification. Although these interface descriptions are intended to be used with the CoRE Link Format, they are applicable for use in any REST interface definition.

The Methods column defines the methods supported by that interface, which are described in more detail below.

Interface	if=	Methods
Link List	core.ll	GET
Batch	core.b	GET, PUT, POST (where applicable)
Linked Batch	core.lb	GET, PUT, POST, DELETE (where applicable)
Sensor	core.s	GET
Parameter	core.p	GET, PUT
Read-only Parameter	core.rp	GET
Actuator	core.a	GET, PUT, POST
Binding	core.bnd	GET, POST, DELETE

The following is an example of links in the CoRE Link Format using these interface descriptions. The resource hierarchy is based on a simple profile defined in Appendix A. These links are used in the subsequent examples below.

```
Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s>;rt="simple.sen";if="core.b",
</s/lt>;rt="simple.sen.lt";if="core.s",
</s/tmp>;rt="simple.sen.tmp";if="core.s";obs,
</s/hum>;rt="simple.sen.hum";if="core.s",
</a>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d>;rt="simple.dev";if="core.ll",
</l>;if="core.lb",
```

5.1. Link List

The Link List interface is used to retrieve (GET) a list of resources on a web server. The GET request SHOULD contain an Accept option with the application/link-format content type, however if the resource does not support any other form of GET methods the Accept option MAY be elided. The Accept option SHOULD only include the application/link-format content type. The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on a web server.

Link List is the base interface to provide gradual reveal of resources on a CoRE web server, hence the root resource of a Function Set SHOULD implement this interface or an extension of this interface.

The following example interacts with a Link List /d containing Parameter sub-resources /d/name, /d/model.

```
Req: GET /d (Accept:application/link-format)
Res: 2.05 Content (application/link-format)
</d/name>;rt="simple.dev.n";if="core.p",
</d/model>;rt="simple.dev.mdl";if="core.rp"
```

5.2. Batch

The Batch interface is used to manipulate a collection of sub-resources at the same time. The Batch interface type supports the same methods as its sub-resources, and can be used to read (GET), set (PUT) or toggle (POST) the values of those sub-resource with a single resource representation. The sub-resources of a Batch MAY be heterogeneous, a method used on the Batch only applies to sub-resources that support it. For example Sensor interfaces do not support PUT, and thus a PUT request to a Sensor member of that Batch would be ignored. A batch requires the use of SenML Media types in order to support multiple sub-resources.

In addition, The Batch interface is an extension of the Link List interface and in consequence MUST support the same methods.

The following example interacts with a Batch /s with Sensor sub-resources /s/light, /s/temp and /s/humidity.

```
Req: GET /s
Res: 2.05 Content (application/senml+json)
{"e":[
    { "n": "light", "v": 123, "u": "lx" },
    { "n": "temp", "v": 27.2, "u": "degC" },
    { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

5.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the web server, a Linked Batch is dynamically controlled by a web client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC5988] and the CoRE Link Format [I-D.ietf-core-link-format]. A request with a POST method and a content type of application/link-format simply appends new resources to the collection. The links in the payload MUST reference a resource on the web server with an absolute path. A DELETE request empties the current collection of links. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /l and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.

```
Req: POST /l (Content-type: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed

Req: GET /l
Res: 2.05 Content (application/senml+json)
{"e":[
   { "n": "/s/light", "v": 123, "u": "lx" },
   { "n": "/s/temp", "v": 27.2, "u": "degC" },
}

Req: POST /l (Content-type: application/link-format)
</s/humidity>
Res: 2.04 Changed

Req: GET /l (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>

Req: GET /l
Res: 2.05 Content (application/senml+json)
{"e":[
   { "n": "/s/light", "v": 123, "u": "lx" },
   { "n": "/s/temp", "v": 27.2, "u": "degC" },
   { "n": "/s/humidity", "v": 80, "u": "%RH" }],
}

Req: DELETE /l
Res: 2.04 Changed
```

5.4. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.

```
Req: GET /s/humidity (Accept: text/plain)
Res: 2.05 Content (text/plain)
80

Req: GET /s/humidity (Accept: application/senml+json)
Res: 2.05 Content (application/senml+json)
{"e":[
    { "n": "humidity", "v": 80, "u": "%RH" }],
}
```

5.5. Parameter

The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or set (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and setting a parameter.

```
Req: GET /d/name
Res: 2.05 Content (text/plain)
node5

Req: PUT /d/name (text/plain)
outdoor
Res: 2.04 Changed
```

5.6. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not set. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

```
Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
```

5.7. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or a new actuator value set (PUT). In addition, this interface defines the use of POST (with no body) to toggle an actuator between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to set.

The following example shows requests for reading, setting and toggling an actuator (turning on a led).

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0

Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed

Req: POST /a/1/led (text/plain)
Res: 2.04 Changed

Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

5.8. Binding

The Binding interface is used to manipulate a binding table. A request with a POST method and a content type of application/link-format simply appends new bindings to the table. All links in the payload MUST have a relation type "boundTo". A GET request simply returns the current state of a binding table whereas a DELETE request empties the table.

The following example shows requests for adding, retrieving and deleting bindings in a binding table.

```
Req: POST /bnd (Content-type: application/link-format)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"
Res: 2.04 Changed

Req: GET /bnd
Res: 2.05 Content (application/senml+json)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"

Req: DELETE /bnd
Res: 2.04 Changed
```

5.9. Resource Observation

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [I-D.ietf-core-observe] MAY be used to observe any changes in a resource, and receive asynchronous notifications as a result. In addition, a set of query string parameters are defined here to allow a client to request how often a client is interested in receiving notifications and how much a resource should change for the new representation to be interesting. These query parameters are described in the following table. A resource using an interface description defined in this specification and marked as Observable in its link description SHOULD support these observation parameters. The Change Step parameter can only be supported on resources with an atomic numeric value.

Query	Parameter	Value
Minimum Period (s)	pmin	xsd:integer (>0)
Maximum Period (s)	pmax	xsd:integer (>0)
Change Step	st	xsd:decimal (>0)

Minimum Period: When present, the minimum period indicates the minimum time in seconds the server SHOULD wait between sending notifications. In the absence of this parameter, the minimum period is up to the server.

Maximum Period: When present, the maximum period indicated the maximum time in seconds the server SHOULD wait between sending notifications (regardless if it has changed). In the absence of this parameter, the maximum period is up to the server. The maximum period MUST be greater than the minimum period parameter

(if present).

Change Step: When present, the change step indicates how much the value of a resource SHOULD change before sending a new notification (compared to the value of the last notification). This parameter has lower priority than the period parameters, thus even if the change step has been fulfilled, the time since the last notification SHOULD be between pmin and pmax.

The following example shows an Observation request using these query parameters. Here the value of Observe indicates the number of seconds since the observation was made to show the time.

```
Req: GET Observe /s/temp?pmin=10&pmax=60&st=1
Res: 2.05 Content Observe:0 (text/plain)
23.2

Res: 2.05 Content Observe:60 (text/plain)
23.0

Res: 2.05 Content Observe:80 (text/plain)
22.0

Res: 2.05 Content Observe:140 (text/plain)
21.8
```

5.10. Future Interfaces

It is expected that further interface descriptions will be defined in this and other specifications. Potential interfaces to be considered for this specifications include:

Collection: This resource would be a container that allows sub-resources to be added or removed.

5.11. WADL Description

This section defines the formal Web Application Description Langauge (WADL) definition of these CoRE interface descriptions.

```
<?xml version="1.0" standalone="yes"?>

<application xmlns="http://research.sun.com/wadl/2006/10"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:senml="urn:ietf:params:xml:ns:senml">
```

```
  <grammars>
    <include href="http://tools.ietf.org/html/draft-jennings-senml"/>
  </grammars>

  <doc title="CoRE Interfaces"/>

  <resource_type id="s">
    <doc title="Sensor resource type"/>
    <method href="#read"/>
    <method href="#observe"/>
  </resource_type>

  <resource_type id="p">
    <doc title="Parameter resource type"/>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#update"/>
  </resource_type>

  <resource_type id="rp">
    <doc title="Read-only Parameter resource type"/>
    <method href="#read"/>
    <method href="#observe"/>
  </resource_type>

  <resource_type id="a">
    <doc title="Actuator resource type"/>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#update"/>
    <method href="#toggle"/>
  </resource_type>

  <resource_type id="ll">
    <doc title="Link List type"/></doc>
    <method href="#listLinks"/>
  </resource_type>

  <resource_type id="b">
    <doc title="Batch of sub-resources type">The methods read,
     observe, update and toggle are applied to each sub-
     resource of the requested resource that supports it. Mixed
     sub-resource types can be supported.</doc>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#update"/>
    <method href="#toggle"/>
    <method href="#listLinks"/>
```

```
  </resource_type>

  <resource_type id="lb">
    <doc title="Linked Batch resource type">. The methods read,
     obervableRead, update and toggle are applied to each linked
     resource of the requested resource that supports it. Mixed
     linked resource types can be supported.</doc>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#update"/>
    <method href="#listLinks"/>
    <method href="#appendLinks"/>
    <method href="#clearLinks"/>
  </resource_type>

  <resource_type id="bnd">
    <doc title="Binding table resource type">A modifiable list of links.
    Each link MUST have the relation type "boundTo".</doc>
    <method href="#listLinks"/>
    <method href="#appendLinks"/>
    <method href="#clearLinks"/>
  </resource_type>

  <method id="read" name="GET">
    <doc>Retrieve the value of a sensor, an actuator or a parameter.
     Both HTTP and CoAP support this method.</doc>
    <request>
    </request>
    <response status="200">
      <representation mediaType="text/plain"/>
      <representation mediaType="application/senml+exi"/>
      <representation mediaType="application/senml+xml"/>
      <representation mediaType="application/senml+json"/>
    </response>
    <response status="2.05">
      <representation mediaType="text/plain"/>
      <representation mediaType="application/senml+exi"/>
      <representation mediaType="application/senml+xml"/>
      <representation mediaType="application/senml+json"/>
    </response>
  </method>

  <method id="observe" name="GET">
    <doc>Observe the value of a sensor, an actuator or a parameter.
     Only CoAP supports this method since it requires the CoRE
     Observe mechanism.</doc>
    <request>
      <param name="pmin" style="query" type="xsd:integer"/>
```

```
      <param name="pmax" style="query" type="xsd:integer"/>
      <param name="st" style="query" type="xsd:decimal"/>
    </request>
    <response status="2.05">
      <representation mediaType="text/plain"/>
      <representation mediaType="application/senml+exi"/>
      <representation mediaType="application/senml+xml"/>
      <representation mediaType="application/senml+json"/>
    </response>
  </method>

  <method id="update" name="PUT">
    <doc>Control the actuator or update a parameter with a new value
    or command. Both HTTP and CoAP support this method.</doc>
    <request>
      <representation mediaType="text/plain"/>
      <representation mediaType="application/senml+exi"/>
      <representation mediaType="application/senml+xml"/>
      <representation mediaType="application/senml+json"/>
    </request>
    <response status="200"/>
    <response status="2.04"/>
  </method>

  <method id="toggle" name="POST">
    <doc>Toggle the values of actuator resources. Both HTTP and CoAP
    support this method.</doc>
    <request>
      <doc>The toggle function is only applicable if the request
      is empty.</doc>
    </request>
    <response status="200"/>
    <response status="2.04"/>
  </method>

  <method id="listLinks" name="GET">
    <doc>Retrieve the list of Web links associated to a resource.
     Both HTTP and CoAP support this method.</doc>
    <request>
    <doc>This request MUST contain an Accept option with
    application/link-format when the resource supports
    other GET methods.</doc>
    </request>
    <response status="200">
      <representation mediaType="application/link-format"/>
    </response>
    <response status="2.05">
      <representation mediaType="application/link-format"/>
```

```
    </response>
  </method>

  <method id="appendLinks" name="POST">
    <doc>Append new Web links to a resource which is a collection
    of links. Both HTTP and CoAP support this method.</doc>
    <request>
      <representation mediaType="application/link-format"/>
    </request>
    <response status="200"/>
    <response status="2.04"/>
  </method>

  <method id="clearLinks" name="DELETE">
    <doc>Clear all Web Links in a resource which is a collection
    of links. Both HTTP and CoAP support this method.</doc>
    <request>
    </request>
    <response status="200"/>
    <response status="2.04"/>
  </method>

</application>
```

6. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this document. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service.

7. IANA Considerations

The interface description types defined require registration.

The new link relation type "boundto" requires registration.

8. Acknowledgments

Acknowledgement is given to colleagues from the SENSEI project who

were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this document.

9. Changelog

Changes from -02 to -03

- Added Bindings
- Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- Defined a Function Set and its guidelines.
- Added the Link List interface.
- Added the Linked Batch interface.
- Improved the WADL interface definition.
- Added a simple profile example.

10. References

10.1. Normative References

[I-D.ietf-core-link-format]
Shelby, Z., "CoRE Link Format", draft-ietf-core-link-format-14 (work in progress), June 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

10.2. Informative References

[I-D.cheshire-dnsext-dns-sd]
Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-11 (work in

progress), December 2011.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-10 (work in progress), June 2012.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-05 (work in progress), March 2012.

[I-D.shelby-core-resource-directory]
Shelby, Z. and S. Krco, "CoRE Resource Directory", draft-shelby-core-resource-directory-03 (work in progress), May 2012.

Appendix A. Profile example

The following is a short definition of simple profile. This simplistic profile is for use in the examples of this document.

Function Set	Root Path	RT	IF
Device Description	/d	simple.dev	core.ll
Sensors	/s	simple.sen	core.b
Actuators	/a	simple.act	core.b

List of Function Sets

Type	Path	RT	IF	Data Type
Name	/d/name	simple.dev.n	core.p	xsd:string
Model	/d/model	simple.dev.mdl	core.rp	xsd:string

Device Description Function Set

Type	Path	RT	IF	Data Type
Light	/s/light	simple.sen.lt	core.s	xsd:decimal (lux)
Humidity	/s/humidity	simple.sen.hum	core.s	xsd:decimal (%RH)

| Temperature | /s/temp | simple.sen.tmp | core.s | xsd:decimal (degC) |

Sensors Function Set

Type	Path	RT	IF	Data Type
LED	/a/{#}/led	simple.act.led	core.a	xsd:boolean

Actuators Function Set

Authors' Addresses

Zach Shelby
Sensinode
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach@sensinode.com

Matthieu Vial
Schneider-Electric
Grenoble,
FRANCE

Phone: +33 (0)47657 6522
Email: matthieu.vial@schneider-electric.com