CoRE
Internet-Draft
Intended status: Informational
Expires: April 21, 2016

Z. Shelby
ARM
M. Vial
Schneider-Electric
M. Koster
ARM

October 19, 2015

Reusable Interface Definitions for Constrained RESTful Environments
draft-ietf-core-interfaces-04

Abstract

This document defines a set of reusable REST resource design patterns suitable for use in constrained environments, based on IETF CoRE standards for information representation and information exchange.

Interface types for Sensors, Actuators, Parameters, and resource Collections are defined using the "if" link attribute defined by CoRE Link Format [RFC6690]. Clients may use the "if" attribute to determine how to consume resources.

Dynamic linking of state updates between resources, either on an endpoint or between endpoints, is defined with the concept of Link Bindings. We also define conditional observation attributes that work with Link Bindings or with simple CoAP Observe [RFC7641].

Copyright Notice

Table of Contents

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces.. CoRE Link-format is a standard for doing Web Linking [RFC5988] in constrained environments. SenML is a simple data model and representation format for composite and complex structured resources. CoRE Link-Format and SenML can be used by CoAP [RFC7252] or HTTP servers.

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. Machine application clients must be able to adapt to different resource organizations without advance knowledge of the specific data structures hosted by each connected thing. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by CoRE Link Format [RFC6690]. CoRE Link Format additionally defines a link attribute for Interface Type ("if") that can be used to describe the REST interface of a resource, and may include a link to a description document.

This document defines a set of Link Format compatible Interface Types for some common design patterns that enable the server side composition and organization, and client side discovery and consumption, of machine resources using Web Linking. An Interface Type may describe a resource in terms of it's associated content formats, data types, URI templates, REST methods, parameters, and responses. Basic interface types are defined for sensors, actuators, and properties. A set of collection types is defined for organizing

resources for discovery, and for various forms of bulk interaction with resource sets using typed embedding links.

This document introduces the concept of a Link Binding, which defines a new link relation type to create a dynamic link between resources over which to exchange state updates. Specifically, a Link Binding is a link for binding the state of 2 resources together such that updates to one are sent over the link to the other. CoRE Link Format representations are used to configure, inspect, and maintain Link Bindings. This document additionally defines a set of conditional Observe Attributes for use with Link Bindings and with the standalone CoRE Observe [RFC7641] method.

Interface Types may be used in the composition of Function Sets and Profiles. Function Sets and Profiles are described and an example is given of a sensor and actuator device profile using Function Sets composed from the Interface Types described in this document.

This document describes a set of Interface Types which are referenced by the "if" link attribute and used to implement reusable design patterns and functional abstractions. A client discovering the "if" link attribute will be able to consume resources based on its knowledge of the expected interface types. In this sense the Interface Type acts in a similar way as a Content-Format, but as a selector for a high level functional abstraction. Interface types may also be provided with hypermedia controls and affordances to drive client interaction using the principles of HATEOAS. In this case, the Interface Types serve as constructor templates for resource organization and hypermedia annotation.

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 [RFC6690]. This specification makes use of the following additional terminology:

Interface Type: A resource attribute which describes the interface exposed by the resource in terms of content formats, REST methods, parameters, and other related characteristics.

Collection: A resource which contains set of related resources, referenced by a list of links and optionally consisting of subresources.

Link Binding: A unidirectional logical link between a source resource and a destination resource, over which state information is synchronized.

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.

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.

3. Interface Types

An Interface Type definition may describe a resource in terms of it’s associated content formats, data types, URI templates, REST methods, parameters, and responses.

4. Collections

4.1. Introduction to Collections

A Collection is a resource which represents one or more related resources. Within this document, a collection refers to a collection with characteristics defined in this document. A Collection Interface Type consists of a set of links and a set of items pointed to by the links which may be sub-resources of the collection resource. The collection types described in this document are Link List, Batch, Linked Batch, and Hypermedia Collection.

The links in a collection are represented in CoRE Link-Format Content-Formats including JSON and CBOR variants, and the items in the collection may be represented by senml, including JSON and CBOR variants. In general, a collection may support items of any available Content-Format.

A particular resource item may be a member of more than one collection at a time by being linked to, but may only be a subresource of one collection.

Some collections may have pre-configured items and links, and some collections may support dynamic creation and removal of items and links. Likewise, modification of items in some collections may be permitted, and not in others.

Collections may support link embedding, which is analogous to an image tag (link) causing the image to display inline in a browser window. Resources pointed to by embedded links in collections may be interacted with using bulk operations on the collection resource. For example, performing a GET on a collection resource may return a single representation containing all of the linked resources.

Links in collections may be selected for processing by a particular request by using Query Filtering as described in CoRE Link-Format [RFC6690].

4.2. Use Cases for Collections

Collections may be used to provide gradual reveal of resources on an endpoint. There may be a small set of links at the .well-known/core location, which may in turn point to other collections of resources that represent device information, device configuration, device management, and various functional clusters of resources on the device.

A collection may provide resource encapsulation, where link embedding may be used to provide a single resource with which a client may interact to obtain a set of related resource values. For example, a collection for manufacturer parameters may consist of manufacturer name, date of manufacture, location of manufacture, and serial number resources which can be read as a single senml data object.

A collection may be used to group a set of like resources for bulk state update or actuation. For example, the brightness control resources of a number of luminaries may be grouped by linking to them in a collection. The collection type may support receiving a single update form a client and sending that update to each resource item in the collection.

Items may be sub-resources of the collection resource. This enables updates to to multiple items in the collection to be processed together within the context of the collection resource.

Items may be dynamically created in a collection along with their hyperlinks. This provides an "item factory" pattern which can serve as a resource creation mechanism for dynamic resources. This pattern is also useful for creating temporary resources for the implementation of dynamic phenomena like commands, actions, and events using REST design patterns. Item creation uses the collection Content-Format which allows specification of links and item state in a single representation.

4.3. Content-Formats for Collections

The collection interfaces by default use CoRE Link-Format for the link representations and SenML or text/plain for representations of items. The examples given are for collections that expose resources and links in these formats. In addition, a new "collection" Content-Format is defined based on the SenML framework which represents both links and items in the collection.

The choice of whether to return a representation of the links or of the items or of the collection format is determined by the accepts header option in the request. Likewise, the choice of updating link metadata or item data or the collection resource itself is determined by the Content-Format option in the header of the update request operation.

The default Content-Formats for collection types described in this document are:

Links: application/link-format, application/link-format+json

Items: application/senml+json, text/plain

Collection: application/collection+senml+json

4.4. Links and Items in Collections

Links use CoRE Link-Format representation by default and may point to any resource reachable from the context of the collection. This includes absolute links and links that point to other network locations if the context of the collection allows. Links to sub-resources in the collection MUST have a path-element starting with the resource name, as per RFC3986 [RFC3986]. Links to resources in the global context MUST start with a root path identifier [RFC5988].Links to other collections are formed per RFC3986.

Examples of links:

</sen/>;if="core.lb" : Link to the /sen/ collection describing it as a core.lb type collection (Linked Batch)

</sen/>;rel="grp" : Link to the /sen/ collection indicating that /sen/ is a member of a group in the collection in which the link appears.

<"/sen/temp">;rt="temperature" : An absolute link to the resource at the path /sen/temp

<temp>;rt="temperature" : Link to the temp subresource of the collection in which this link appears.

<temp>;anchor="/sen/" : A link to the temp subresource of the collection /sen/ which is assumed not to be a subresource of the collection in which the link appears ,but is expected to be identified in the collection by resource name.

Links in the collection MAY be Read, Updated, Added, or Removed using the CoRE Link-Format or JSON Merge-Patch Content-Formats on the collection resource. Reading links uses the GET method and returns an array or list containing the link-values of all selected links. Links may be added to the collection using POST or PATCH methods. Updates to links MUST use the PATCH method and MAY use query filtering to select links for updating. The PATCH method on links MUST use the JSON Merge-Patch Content-Format (application/merge-patch+json) specified in RFC7396 [RFC7396] .

Items in the collection SHOULD be represented using the SenML (application/senml+json) or plain text (text/plain) Content-Formats, depending on whether the representation is of a single data point or multiple data points. Items MAY be represented using any supported Content-Format.

Link Embedding enables the bulk processing of items in the collection using a single operation targeting the collection resource. A subset of resources in the collection may be selected for operation using Query Filtering. Bulk Read operations using GET return a SenML representation of all selected resources. Bulk item Update operations using PUT or POST apply the payload document to all selected resource items in the collection, using a either a Batch or Group update policy. A Batch update is performed by applying the resource values in the payload document to all resources in the collection that match any resource name in the payload document. Group updates are performed by applying the payload document to each item in the collection. Group updates are indicated by the link relation type rel="grp" in the link.

The collection resource SHOULD represented using the collection+senml+json Content-Format. The Hypermedia Collection type is the only collection type which supports this representation. Reading a collection using this content-format returns a representation of the links and the items in the collection. Performing a POST operation using this Content-Format MAY create one or more new item(s) and their corresponding links in the collection. Performing a PUT operation on this resource replaces the entire set of links and items with the payload. This Content-Format is described in section Section 4.9. Implementations MAY provide an alternate method using POST in a Content-Format used by the items in the collection which creates a default link-value and system-assigned resource name. Such implementations MAY create sub-resources of the collection resource.

4.5. Queries on Collections

Collections MAY support query filtering as defined in CoRE Link-Format [RFC6690]. Operations targeting either the links or the items MAY select a subset of links and items in the collection by using query filtering. The Content-Format specified in the request header selects whether links or items are targeted by the operation.

4.6. Observing Collections

Resource Observation using CoAP [RFC7252] MAY be supported on items in a collection. A subset of the conditional observe parameters MAY be specified to apply. In most cases pmin and pmax are useful. Resource observation on a collection’s items resource MAY report any changes of resource state in any item in the collection. Observation Responses, or notifications, SHOULD provide representations of the resources that have changed in SenML Content-Format. Notifications MAY include multiple observations of a particular resource, with SenML time stamps indicating the observation times.

4.7. Hypermedia Controls on Collections

Additional Hypermedia controls may be defined to enable clients to automatically consume the collection resources. Typically, the developer may map application level semantics onto collection operations. For example, invoking an Action on an actuator may be defined as creating an Action item resource in a collection of Actions associated with the actuator, each item in the collection representing a past, current, or future action to be processed by the actuator. Removing the item could cancel any pending or curent long-running action, and removing a completed action could free up resources for new actions to be invoked. A Hypermedia control for this pattern might provide a semantic name for the action, for

example "Change Brightness", and might direct the client to supply a SenML representation of parameters for the action as well as provide instructions on what method (POST) to use and how to construct the URI (the collection URI in this case) if required. An example of this hypermedia control is shown below.

4.8. Collection Types

There are four collection types defined in this document:

Collection Type	if=	Content-Formats
Link List	core.ll	link-format
Batch	core.b	link-format, senml
Linked Batch	core.lb	link-format, senml
Hypermedia Collection	core.hc	link-format, senml, collection+senml
Binding	core.bnd	link-format

Each collection type MAY support a subset of the methods and functions described above. For the first three collection types, the methods and functions are defined in the corresponding Interface Description. The Hypermedia Collection SHOULD expose hypermedia controls to applications to indicate which methods and functions are supported.

4.9. The collection+senml+json Content-Format

The collection+senml+json Content-Format is used to represent all of the attributes and resources of a collection in a single format. This is accomplished by extending the SenmL format by adding a links element "l". The links element is formatted as an array of links in the application/link-format+json Content-Format with the tag "l" which follows the structure of the "e" element. An example of this format is given below.

```
{
  "bn":"/ep/sen/"
  "e":[
    { "n": "light", "v": 123, "u": "lx" },
    { "n": "temp", "v": 27.2, "u": "degC" },
    { "n": "humidity", "v": 80, "u": "%RH" }],
  "l":[
    { "href":"/ep/sen/", "rel":"self", "if": "core.hc", "rt": "ms" },
    { "href":"light", "rt":"core.s" },
    { "href":"temp", "rt":"core.s" },
    { "href":"humidity", "rt":"core.s" }]
}
```

5. Link Bindings and Observe Attributes

In a M2M RESTful environment, endpoints may directly exchange the content of their resources to operate the distributed system. For example, a light switch may supply on-off control information that may be sent directly to a light resource for on-off control. 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 link 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 5.2.

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

5.1. Format

Since Binding involves 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)
Greater Than	gt	xsd:decimal
Less Than	lt	xsd:decimal

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 state synchronization 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.

Greater Than: When present, Greater Than indicates the upper limit value the resource value SHOULD cross before sending a new

notification. This parameter has lower priority than the period parameters, thus even if the Greater Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

Less Than: When present, Less Than indicates the lower limit value the resource value SHOULD cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Less Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

5.2. 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, Less Than, Greater Than).

Observe: The Observe method creates an observation relationship 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 [RFC7641]

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 5.4).

Push: When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the binding condition attributes are satisfied for 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.

5.3. 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 6.9. A profile SHOULD allow only one resource table per endpoint.

5.4. Resource Observation Attributes

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [RFC7641] 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 control how often a client is interested in receiving notifications and how much a resource value 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.

These query parameters MUST be treated as resources that are read using GET and updated using PUT, and MUST NOT be included in the Observe request. Multiple parameters MAY be updated at the same time by including the values in the query string of a PUT. Before being updated, these parameters have no default value.

Resource	Parameter	Data Format
Minimum Period	/{resource}?pmin	xsd:integer (>0)
Maximum Period	/{resource}?pmax	xsd:integer (>0)
Change Step	/{resource}?st	xsd:decimal (>0)
Less Than	/{resource}?lt	xsd:decimal
Greater Than	/{resource}?gt	xsd:decimal

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 state synchronization 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.

Greater Than: When present, Greater Than indicates the upper limit value the resource value SHOULD cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Greater Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

Less Than: When present, Less Than indicates the lower limit value the resource value SHOULD cross before sending a new notification. This parameter has lower priority than the period parameters, thus even if the Less Than limit has been crossed, the time since the last notification SHOULD be between pmin and pmax.

6. 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	Content-Formats
Link List	core.ll	GET	link-format
Batch	core.b	GET, PUT, POST	link-format, senml
Linked Batch	core.lb	GET, PUT, POST, DELETE	link-format, senml
Sensor	core.s	GET	link-format, text/plain
Parameter	core.p	GET, PUT	link-format, text/plain
Read-only Parameter	core.rp	GET	link-format, text/plain
Actuator	core.a	GET, PUT, POST	link-format, text/plain
Binding	core.bnd	GET, POST, DELETE	link-format

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",
```

6.1. 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 format; 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 format. 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"
```

6.2. 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), update (PUT) or apply (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" }],
}
```

6.3. 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 [RFC6690]. A request with a POST method and a content format of application/link-format simply appends new resource links to the collection. The links in the payload MUST reference a resource on the web server with an absolute path. A DELETE request removes the entire collection. 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-Format: 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-Format: 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: 2.02 Deleted
```

6.4. Hypermedia Collection

The Hypermedia Collection interface MAY provide a full set of the methods and link relation types described in section Section 4 of this document.

The following example interacts with a Hypermedia Collection at /act1/actions/ by creating a new resource with Parameter sub-resources newVal, tTime. The example depicts an actuation operation with a new actuator value of 86.3% and a transition time of 10 seconds. The returned location of the created resource is then read, and a response is returned which includes the remaining time for the operation to complete "rTime". Then, the operation is cancelled by sending a DELETE operation to the location of the created resource that represents the running action.

```
Req: POST /Act1/Actions/
Content-Format: application/collection+senml_json
Pl: [{"n":newVal", "v":86.3}, {"n":tTime", "v":10}]
Res: 2.01 Created
Location: Action1234

Req: GET /Act1/Actions/Action1234
Accepts: application/senml+json
Res: 2.05 Content
Pl: [{"n":newVal", "v":86.3},
     {"n":tTime", "v":10},
     {"n":"rTime", "v":"8.87"}]

Req: DELETE /Act1/Actions/Action1234
Res: 2.02 Deleted

Req: GET /Act1/Actions/Action1234
Res: 4.04 Not Found
```

6.5. 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" }],
}
```

6.6. 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 update (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

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

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

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

6.7. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not updated. 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
```

6.8. 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 the actuator value can be updated (PUT). In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle 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 updated.

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
```

6.9. Binding

The Binding interface is used to manipulate a binding table. A request with a POST method and a content format 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-Format: 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/link-format)
<coap://sensor.example.com/s/light>;
  rel="boundto";anchor="/a/light";bind="obs";pmin="10";pmax="60"

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

6.10. Future Interfaces

It is expected that further interface descriptions will be defined in this and other specifications.

6.11. 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 interface type"/>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#observe-cancel"/>
    <method href="#getattr"/>
    <method href="#setattr"/>
  </resource_type>

  <resource_type id="p">
    <doc title="Parameter interfacee type"/>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#observe-cancel"/>
    <method href="#getattr"/>
    <method href="#setattr"/>
    <method href="#update"/>
  </resource_type>

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

```
    <method href="#setattr"/>
  </resource_type>

  <resource_type id="a">
    <doc title="Actuator interface type"/>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#observe-cancel"/>
    <method href="#getattr"/>
    <method href="#setattr"/>
    <method href="#update"/>
    <method href="#apply"/>
  </resource_type>

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

  <resource_type id="b">
    <doc title="Batch of sub-resources interface type">The methods read,
     observe, update and apply 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="#observe-cancel"/>
    <method href="#getattr"/>
    <method href="#setattr"/>
    <method href="#update"/>
    <method href="#apply"/>
    <method href="#listLinks"/>
  </resource_type>

  <resource_type id="lb">
    <doc title="Linked Batch interface type">. The methods read,
     obervableRead, update and apply 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="#observe-cancel"/>
    <method href="#getattr"/>
    <method href="#setattr"/>
    <method href="#update"/>
    <method href="#apply"/>
    <method href="#listLinks"/>
    <method href="#appendLinks"/>
```

```
    <method href="#clearLinks"/>
  </resource_type>

  <resource_type id="hc">
    <doc title="Hypermedia Collection interface type">.</doc>
    <method href="#read"/>
    <method href="#observe"/>
    <method href="#observe-cancel"/>
    <method href="#getattr"/>
    <method href="#setattr"/>
    <method href="#update"/>
    <method href="#apply"/>
    <method href="#listLinks"/>
    <method href="#appendLinks"/>
    <method href="#clearLinks"/>
    <method href="#updateLinks"/>
    <method href="#readCollection"/>
    <method href="#addItem"/>
  </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="observe" style="header" type="xsd:integer">
        <option value = 0/>
      </param>
    </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="observe-cancel" name="GET">
    <doc>Cancel observation in progress.
     Only CoAP supports this method since it requires the CoRE
     Observe mechanism.</doc>
    <request>
      <param name="observe" style="header" type="xsd:integer">
        <option value = 1/>
      </param>
    </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="getattr" name="GET">
```

```
    <doc>Retrieve the observe attributes associated with 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>
      <representation mediaType="application/link-format"/>
    </request>
    <response status="200">
      <representation mediaType="application/link-format"/>
    </response>
    <response status="2.05">
      <representation mediaType="application/link-format"/>
    </response>
  </method>

  <method id="setattr" name="PUT">
    <doc>Set the values of some or all of the observe attributes
    associated with a resource.
    Both HTTP and CoAP support this method.</doc>
    <request>
      <param name="pmin" style="query" type="xsd:integer"/>
      <param name="pmax" style="query" type="xsd:integer"/>
      <param name="lt" style="query" type="xsd:decimal"/>
      <param name="gt" style="query" type="xsd:decimal"/>
      <param name="st" style="query" type="xsd:decimal"/>
    </request>
    <response status="200">
    </response>
    <response status="2.04">
    </response>
  </method>

  <method id="apply" name="POST">
    <doc>Apply the value, if supplied, to resources. Both HTTP and CoAP
    support this method.</doc>
    <request>
      <doc>The apply function may contain a payload to be applied.</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="2.02"/>
  </method>

  <method id="updateLinks" name="PATCH">
    <doc>Update all Web Links in a resource which is a collection
    of links. Both HTTP and CoAP support this method.</doc>
    <doc>This request MUST contain a Content-Format option with
    application/merge-patch+json.</doc>
    <request>
    </request>
    <response status="200"/>
    <response status="2.04"/>
  </method>

    <method id="addItem" name="POST">
    <doc>Add zero or more items to the collection with their links. Both HTT
P and CoAP support this method.</doc>
    <doc>This request MAY contain a Content-Format option with
    application/collection+senml+json.</doc>
    <doc>This request MAY contain a Content-Format option with
    application/senml+json.</doc>
    <request>
```

```
    </request>
    <response status="200"/>
    <response status="2.01"/>
  </method>

    <method id="readCollection" name="GET">
    <doc>REturn a representation of both links and items in the collection.
Both HTTP and CoAP support this method.</doc>
    <doc>This request MUST contain an Accepts option with
    application/collection+senml+json.</doc>
   <request>
    </request>
    <response status="200"/>
    <response status="2.05"/>
  </method>

</application>
```

7. Function Sets and Profiles

This section defines how a set of REST resources can be created called a function set. 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 can have a subset of mandatory inputs, outputs and parameters to provide minimum interoperability. It can also be extended with manufacturer/user-specific resources. A device is composed of one or more Function Set instances.

An example of function sets can be found from the CoRE Resource Directory specification that defines REST interfaces for registration, group and lookup [I-D.ietf-core-resource-directory]. The OMA Lightweight M2M standard [REF] also defines a function set structure called an Objects that use integer path, instance and resource URI segments. OMA Objects can be defined and then registered with an OMA maintained registry [REF]. This section is simply meant as a guideline for the definition of other such REST interfaces, either custom or part of other specifications.

7.1. 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.

7.1.1. 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 can 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 should be fixed.

7.1.2. 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 can be exported to DNS-SD [RFC6763] for example.

The Resource Type parameter defines the value that should 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 could allow mixing in foreign namespace references within a Function Set to import external references from other object models (e.g. SenML and UCUM).

7.1.3. Interface Description

The Interface Description parameter defines the REST interface for that type of resource. Several base interfaces are defined in Section 6 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 6.1 offers this functionality so a root resource should support this interface or a derived interface like CoRE Batch (See Section 6.2).

7.1.4. 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 6 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 formats for the CoRE interfaces or define new interfaces with new content types.

7.2. 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 [RFC6690]. 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.ietf-core-resource-directory] if such a directory is available.

7.3. Versioning

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

8. 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.

9. IANA Considerations

The interface description types defined require registration.

The new link relations type "boundto" and "grp" require registration.

10. 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.

11. Changelog

Changes from -03 to -04

- Fixed tickets #385 and #386
- Changed abstract and into to better describe content
- Focus on Interface and not function set/profiles in intro
- Changed references from draft-core-observe to RFC7641
- Moved Function sets and Profiles to section after Interfaces
- Moved Observe Attributes to the Link Binding section
- Add a Collection section to describe the collection types
- Add the Hypermedia Collection Interface Description

Changes from -02 to -03

- Added lt and gt to binding format section.
- Added pmin and pmax observe parameters to Observation Attributes
- Changed the definition of lt and gt to limit crossing.
- Added definitions for getattr and setattr to WADL.
- Added getattr and setattr to observable interfaces.

- Removed query parameters from Observe definition.
- Added observe-cancel definition to WADL and to observable interfaces.

Changes from -01 to -02

- Updated the date and version, fixed references.
- Removed pmin and pmax observe parameters [Ticket #336]

Changes from -00 to WG Document -01

- Improvements to the Function Set section.

Changes from -05 to WG Document -00

- Updated the date and version.

Changes from -04 to -05

- Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.

Changes from -03 to -04

- Draft refresh

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.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

12.2. Informative References

[I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-04 (work in progress), July 2015.

[I-D.jennings-core-senml] Jennings, C., Shelby, Z., Arkko, J., and A. Keranen, "Media Types for Sensor Markup Language (SENML)", draft-jennings-core-senml-01 (work in progress), July 2015.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <http://www.rfc-editor.org/info/rfc6763>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <http://www.rfc-editor.org/info/rfc7396>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

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
ARM
150 Rose Orchard
San Jose 95134
FINLAND

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Matthieu Vial
Schneider-Electric
Grenoble
FRANCE

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

Michael Koster
ARM
150 Rose Orchard
San Jose 95134
USA

Email: michael.koster@arm.com