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

M. Vial
Schneider-Electric

April 10, 2013

CoRE Mirror Server
draft-vial-core-mirror-server-01

Abstract

The Constrained RESTful Environments (CoRE) working group aims at realizing the REpresentational State Transfer (REST) architecture in a suitable form for the most constrained nodes. Thanks to the Constrained Application Protocol (CoAP), REST is now applicable to constrained networks. However the most energy-constrained devices may enter sleep mode and disconnect their network link during several minutes to save energy, hence preventing them from acting as traditional web servers. This document describes how a sleeping device can store resource representations in an entity called Mirror Server and participate in a constrained RESTful environment.

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 October 12, 2013.

Copyright Notice

Table of Contents

1. Introduction

1.1. Motivation

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, energy harvesting) and networks (e.g. 6LoWPAN). The CoRE charter says that the CoAP protocol [I-D.ietf-core-coap] will support various form of "caching" to support sleeping devices. And the CoAP requirement REQ3 in [I-D.shelby-core-coap-req] clearly states that

support of sleeping devices is required:

> The ability to deal with sleeping nodes. Devices may be powered off at any point in time but periodically "wake up" for brief periods of time.

As pointed out by [I-D.arkko-core-sleepy-sensors], the server model is not appropriate for the most energy-constrained devices. CoAP also supports the Publish/Subscribe pattern through CoAP observe [I-D.ietf-core-observe]. Notifications with CoAP observe prove to be efficient however as it is currently specified, it still requires the server model to create and maintain the observation relationship. Although CoAP observe may be enhanced to support subscriptions initiated by the observed server, this method is not currently specified. Also in general, a SEP would support only a limited number of observers at a time. The client model is a viable approach but the interactions and interfaces between endpoints are currently undefined. In conclusion, the current working group documents do not propose a complete solution for sleeping devices that are not always reachable.

1.2. Uses cases

With the emergence of the Internet of Things we expect a major breaktrough in the number of smart objects in our environment. Yet providing these objects with sufficient energy for continued operation and long battery lifetime is still a big challenge. That is the reason why this specification strives to provide a solution to dramatically reduce the power consumption of constrained RESTful sensors. For battery-operated devices the need to improve battery lifetime is persistent either to reduce the size of smart objects and fit new applications, to increase the product lifetime when it is directly coupled to its battery lifetime or to reduce the annoyance, costs and wastes incurred by changing batteries too frequently. There is also a new trend to avoid batteries and create sensors that can harvest energy from their environment. For those devices it is of prime importance to maintain a high ratio between harvested energy and power consumption. This ratio has a direct impact on service availability and the user experience especially because the harvesting efficiency is typically not constant in time (e.g day/ night for a photovoltaic cell).

1.3. Assumptions and objectives

In this specification we assume that the energy-constrained devices can store a sufficient amount of energy to enable bi-directional communication and to perform periodic tasks like maintaining soft state. However the most constrained devices may not be able to store energy and may have unpredictable availability due to sporadic energy production (e.g. self-powered push button). This specification may be applicable to these devices as long as they have enough energy to perform the initial registration. This may require an additional source of power during the commissioning phase.

Throughout this document we will only consider sleeping devices that are totally unreachable during long periods of time. In other word, network connectivity is turned off at least several seconds hence generating unacceptable interruptions if the device runs as a server. Some link-layer technologies offer advanced low power modes such as duty-cycle link activity or receiver initiated transmissions hence allowing the devices to sleep while still offering network connectivity with an always-on illusion. Devices for which the available energy is sufficient to afford always-on illusion are out of scope of this specification since the server model is applicable to these endpoints.

Efficient support of sleeping devices has implications on many aspects of the IP stack: Media Access Control (MAC), neighbor discovery, routing, REST intermediaries... This specification does not aim to find a solution for all of those. The objective is to provide an interaction model at the application level where data exchanges are always initiated by the sleeping endpoint. This way the application can finely control when the network link needs to be on. In no way the mechanisms defined here precludes usage of a low power mode at link-layer.

This specification does not pretend to provide full REST support to sleeping devices. These devices will be provided with the minimum set of REST features to publish resources. Particular attention is paid to facilitate configuration and to associate meta-data to resources from sleeping devices.

2. Requirements Language

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 RFC 2119 [RFC2119].

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC5988], [I-D.shelby-core-resource-directory] and [I-D.shelby-core-interfaces]. Readers should also be familiar with the terms and concepts discussed in [I-D.ietf-core-coap] and [I-D.ietf-core-link-format]. This specification makes use of the following additional terminology:

Sleeping device: A smart object that can enter a long period of time with its network link in disconnected state in order to save energy.

Sleeping endpoint (SEP): A sleeping endpoint is an IP sleeping device which can participate in a constrained RESTful environment as a client-only endpoint.

Mirror Server (MS): A server endpoint that implements the Mirror Server Function Set.

3. Architecture

The Mirror Server architecture is shown in Figure 1. A Mirror Server (MS) is a web server implementing a special Function Set that allows a sleeping endpoint (SEP) to create resources in the MS resource tree. For energy efficiency a SEP is a client-only CoAP endpoint and hence is not able to serve content by itself. The MS implements REST interfaces allowing a SEP to maintain a set of mirrored resources that will be served in turn by the MS. So a Mirror Server acts as a mailbox between the sleeping endpoint and the client. A CoAP client discovers resources owned by the SEP but hosted on the MS using typical mechanisms such as "/.well-known/core" [I-D.ietf-core-link-format] or Resource Directory [I-D.shelby-core-resource-directory].

A SEP must register and maintain a mirror entry on the MS, which is soft state and need to be periodically refreshed. A MS provides interfaces to register, update and remove a mirror entry and an associated set of mirrored resources. Furthermore, a MS provides interfaces to read and update the mirrored resources from both the SEP and client sides. Finally, a mechanism to discover a MS using the CoRE Link Format is defined.

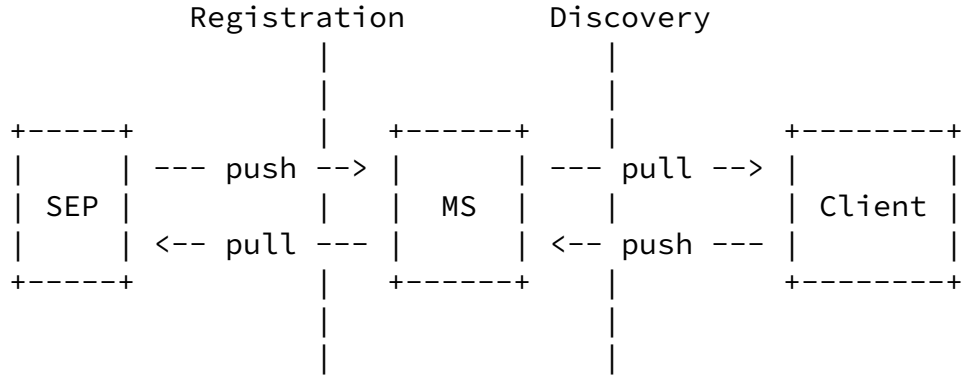


Figure 1: Mirror Server architecture

The Mirror Server functionality can be distributed over multiple server endpoints in the network or centralized on a single server endpoint. A shorter round-trip time gives better energy efficiency for request/response exchanges, so it is important to choose a path between the Mirror Server and the sleeping endpoint with minimum

latency. Moreover a sleeping endpoint with a Mirror Server in its direct neighborhood may even avoid having to configure global IP connectivity. However in a wireless network relying on local connectivity may result in fragility due to device mobility or radio fluctuations. This could lead a sleeping endpoint to frequently try to move from one Mirror Server to another. Consequently, clients would need to restart resource discovery frequenlty. In that regard, a centralized Mirror Server gives more stability. A centralized Mirror Server also concentrates network traffic on a central point and may cause network congestion in a mesh network. However data flow of a sleeping endpoint is expected to be low hence mitigating the risk of network congestion.

A sleeping endpoint MAY register with more than one Mirror Server but in that case the resources of a sleeping endpoint appear duplicated

during resource discovery. Section 4.1 describes how to detect duplicate resources.

4. Mirror Server Function Set

The interface is mostly identical to that of the Resource Directory Function Set defined in [I-D.shelby-core-resource-directory] so this specification only points out the differences. Contrary to the Resource Directory there is no lookup Function Set in a Mirror Server. Indeed, from a client point of view, the mirrored resources look like any other resources hosted the MS endpoint. So resource discovery of mirrored resources is directly available through "/.well-known/core" instead of a separate Function Set.

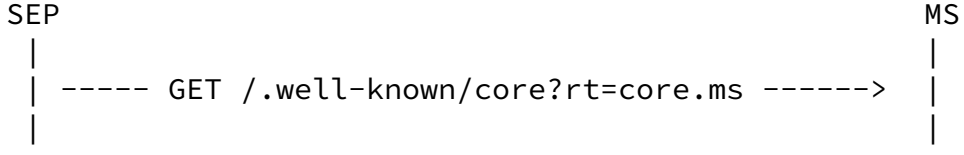
The examples presented in this section make use of a smart temperature sensor the resources of which are defined below using Link Format. Three resources are dedicated to the Device Description (manufacturer, model, name) and one contains the current temperature in degree Celsius.

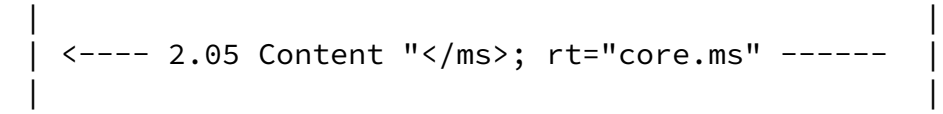
```
</dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</dev/n>;rt="ipso.dev.n";if="core.p",
</sen/temp>;rt="ucum.Cel";if="core.s";obs
```

4.1. Discovery

The interaction between a SEP and a MS is based on the same discovery interface as the Resource Directory except that the Resource Type in the URI template is replaced with "core.ms".

The following example shows a sleeping endpoint discovering a MS using this interface, thus learning that the base MS resource is at /ms.

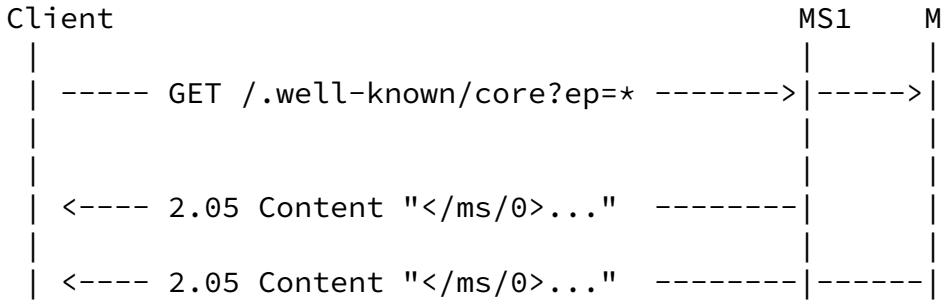




Req: GET coap://[ff02::1]/.well-known/core?rt=core.ms
Res: 2.05 Content
</ms>;rt="core.ms"

Resource discovery between a client and a MS or a client and a RD needs special care to take into account the fact that resources from a sleeping endpoint might appear duplicated. Clients SHOULD employ 2-step resource discovery by looking up sleeping endpoints AND resource types to detect duplicate resources. Clients MAY use single-step resource discovery only if the SEP can register with no more than one Mirror Server. A client can use the "ep" link attribute as a filter on the "/.well-known/core" resource to retrieve a list of endpoints and detect duplicate sleeping endpoints registered on multiple MSs. A client can use the "ep" type of lookup to do the same on a RD. The result of endpoint discovery is then used to filter out duplicate resources returned from simple resource discovery.

The following example shows a client discovering the sleeping endpoints and learning that the SEP 0224e8fffe925dcf is registered on two Mirror Servers.

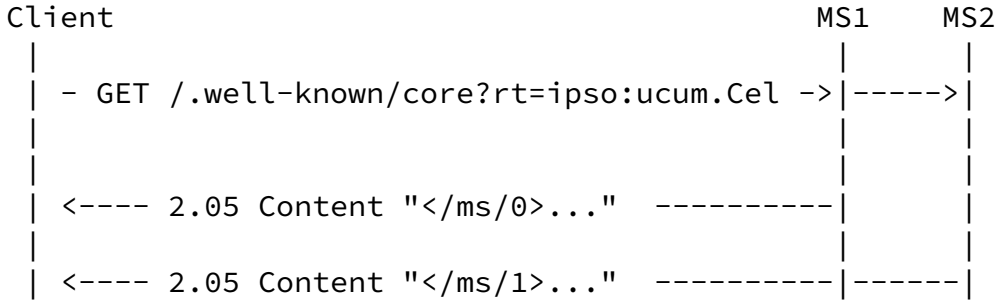


Req: GET coap://[ff02::1]/.well-known/core?ep=*
Res: 2.05 Content
</ms/0>;ep="0224e8fffe925dcf"
Res: 2.05 Content

</ms/0>;ep="02004cfffe4f4f50"

</ms/1>;ep="0224e8fffe925dcf"

From the previous exchange and the next resource discovery request, the client can infer that the resources coap://ms1/ms/0/sen/temp and coap://ms2/ms/1/sen/temp actually come from the same sleeping endpoint.



Req: GET coap://[ff02::1]/.well-known/core?rt=ucum.Cel
Res: 2.05 Content
</ms/0/sen/temp;rt="ucum.Cel"
Res: 2.05 Content
</ms/1/sen/temp>;rt="ucum.Cel"

4.2. Registration

The registration interface is identical to the registration interface of the Resource Directory Function Set except that the preferred path for the Mirror Server Function Set is "/ms".

After discovering the location of a MS Function Set, a sleeping endpoint MAY register its resources that need to be mirrored using the registration interface. This interface accepts a POST from an endpoint containing a description of the resources to be created on the Mirror Server as the message payload in the CoRE Link Format along with query string parameters indicating the endpoint identifier, its domain and the lifetime of the registration. The Link Format description is identical to the "/.well-known/core" resource found on a typical server endpoint meaning that the Interface Description attributes are actually intended for the Mirror Server. A Mirror Server MUST reject a registration if at least one of the Interface Descriptions is not supported. Upon successful registration a MS creates a new resource or updates an existing resource for the mirror entry and returns its location. The resources specified by the SEP during registration are created as sub-resources of the mirror entry on the MS endpoint. The registration interface MUST be implemented to be idempotent, so that

registering twice with the same endpoint parameter does not create multiple MS entries. The resource associated to a mirror entry SHOULD implement the Interface Type CoRE Link List defined in [I-D.shelby-core-interfaces]. A GET request on this resource MUST return the list of mirrored resources for the corresponding SEP.

After successul registration, a MS SHOULD enable resource discovery for the new mirrored resources by updating its "/.well-known/core" resource. A MS MUST wait for the initial representation of a mirrored resource before it can be visible in resource discovery. The top level resource corresponding to a mirror entry MUST be published in "/.well-known/core" to enable 2-step resource discovery described in Section 4.1. Sub-resources of a mirror entry SHOULD be discoverable either directly in "/.well-known/core" or indirectly through gradual reveal from the mirror entry resource. The Web Link of a mirror entry MUST contain an "ep" attribute with the value of the End-Point parameter received at registration. If present, the End-Point Type parameter SHOULD also be mapped as a "rt" attribute.

A Mirror Server MAY be configured to register the SEP's resources in a Resource Directory (RD). A SEP MUST NOT register the mirrored resources in a RD by itself. It is always the responsibility of the Mirror Server. Since each SEP may register resources with different lifetimes, a MS MUST register the resources of a SEP in a separate resource directory entry. A SEP may register with multiple MS hence the RD entries from the different MS for the same SEP would overlap if special care is not taken. Therefore if a SEP is likely to register with more than one MS, a Mirror Server MUST create its own domain to register the resources of a SEP. This precaution ensures that the ep identifier of a SEP is unique for each domain in the RD. The new domain is typically formed by concatenating the MS's endpoint identifier with the domain in use.

SEP resources in the MS are kept active for the period indicated by the lifetime parameter. The SEP is responsible for refreshing the entry within this period using either the registration or update interface. Once a mirror entry has expired, the MS deletes all resources associated to that entry and updates its "/.well-known/core" resource. When the mirrored resources are also registered in a RD, the RD and MS entries are supposed to have the same lifetime. Consequently, when the mirror entry expires, a MS MAY let the RD entry expire too instead of explicitly deleting it. Nevertheless if the MS entry is deleted using the Removal interface then the RD entry MUST be explicitly removed.

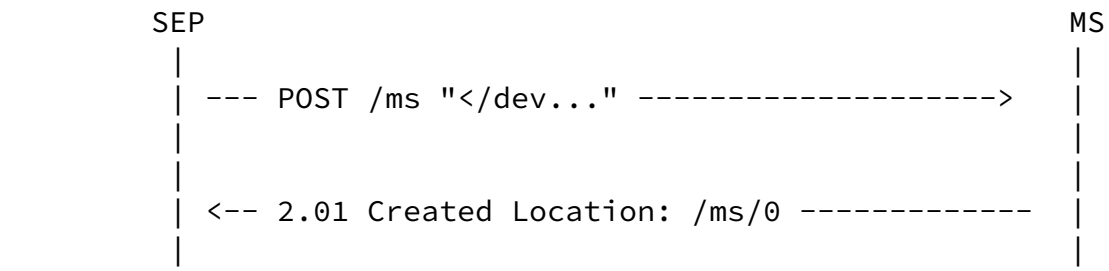
A Mirror Server could lose or delete the mirror entry associated to a

SEP without sending an explicit notification (e.g. after reboot). A SEP SHOULD be able to detect this situation by processing the

response code while using the SEP Operation or Update interface. Especially an error code "4.04 Not Found" SHOULD cause the SEP to register again. A SEP MAY also register with multiple MSs to alleviate the risk of interruption of service.

Implementation note: It is not recommended to reuse the value of the ep parameter in the URI of the Mirror Server entry. This parameter may be a relatively long identifier to guarantee global uniqueness (e.g. EUI64) and would generate inefficient URIs on the Mirror Server where only a local handler is necessary.

The following example shows a sleeping endpoint registering with a MS.




```
Req: POST coap://ms.example.org/ms?ep=0224e8fffe925dcf&rt=sensor
Etag: 0x3f
Payload:
</dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</dev/n>;rt="ipso.dev.n";if="core.p",
</sen/temp>;rt="ucum.Cel";if="core.s";obs

Res: 2.01 Created
Location: /ms/0
```

The mirror entry resource below has been created on the MS.

```
Req: GET coap://ms.example.org/.well-known/core
Res: 2.05 Content
</ms>;rt="core.ms",
```

```
</ms/0>;ep="0224e8fffe925dcf";rt="sensor";if="core.ll"
```

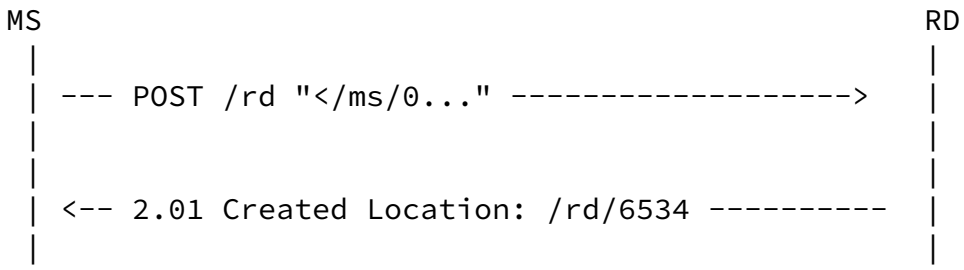
The SEP sets the initial value for its mirrored resources and the following resources are now created.

```
Req: GET coap://ms.example.org/ms/0
Res: 2.05 Content
Payload:
```



```
</ms/0/dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</ms/0/dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</ms/0/dev/n>;rt="ipso.dev.n";if="core.p",
</ms/0/sen/temp>;rt="ucum.Cel";if="core.s";obs
```

Then the MS registers the mirrored resources in the RD.




```
Req: POST coap://rd.example.org/rd?ep=0224e8fffe925dcf&
                  rt=sensor&d=ms1.example.org
Etag: 0x6a
Payload:
</ms/0/dev/mfg >;rt="ipso.dev.mfg";if="core.rp",
</ms/0/dev/mdl>;rt="ipso.dev.mdl";if="core.rp",
</ms/0/dev/n>;rt="ipso.dev.n";if="core.p",
</ms/0/sen/temp>;rt="ucum.Cel";if="core.s";obs

Res: 2.01 Created
Location: /rd/6534
```

4.3. Update

The update interface is not necessary on a Mirror Server. A SEP can

use the registration interface to modify a mirror entry. The Lifetime query parameter of the SEP operation interface defined in Section 4.6 allows a SEP to extend the lifetime of its mirror entry.

4.4. Validation

The validation interface is not supported on a Mirror Server since the sleeping endpoint is unreachable.

4.5. Removal

The removal interface is identical.

Upon successful removal, "/.well-known/core" and the Resource Directory (if applicable) MUST be updated accordingly. All resources associated to the mirror entry are deleted.

4.6. SEP Operation

The SEP Operation interface is not defined for a Resource Directory and is specific to the Mirror Server Function Set.

Once the resources have been created on the MS, the SEP can access its mirrored resources at its own pace. The SEP MAY update its mirrored resources on the MS using PUT requests. The SEP MAY retrieve the current representation of any of its mirrored resources using GET requests. The SEP can reactivate its mirror entry by including a Lifetime (lt) parameter in the query string. While updating dynamic resources, a SEP SHOULD include a Lifetime parameter with the smallest value that matches its technical constraints. It allows a client to fastly detect a stale mirror entry. A SEP MAY omit processing some responses for non confirmable requests in order to avoid spending energy waiting for a response when it is frequently accessing a mirrored resource. Nevertheless a SEP SHOULD periodically check the responses to ensure that its mirror entry is still active on the MS.

Other specifications may override or extend this interface to provide more advanced features, support other REST methods and queuing patterns. This is however out of scope of this specification, which provides only a basic behavior.

In addition to the Modification check interface defined in Section 4.8, the SEP Operation interface support a lightweight mechanism to detect if a mirrored resource has been changed by a client. When a SEP sends a PUT request to update its resources, the response may contain a link-format payload. The payload does not directly relate to the target resource of the PUT request. Instead, it is a list of web links to resources that have been modified by clients since either the last PUT request or the last call to the modification check interface.

The basic SEP operation interface is specified as follows:

Interaction: SEP -> MS

Method: GET, PUT

URI Template: /{+location}{+resource}{?lt}

URI Template Variables:

location := This is the Location path returned by the MS as a result of a successful registration.

resource := This is the relative path to a mirrored resource managed by the registered SEP.

lt := Lifetime (optional). The number of seconds by which the lifetime of the whole mirror entry is extended. Range of 1-4294967295. If no lifetime is included, the current remaining lifetime stays unchanged.

Request Content-Type: Defined at registration

Response Content-Type: Defined at registration for GET method. application/link-format for PUT method if at least one of the mutable resources has been updated since the last PUT request.

Etag: The Etag option MAY be included to allow clients to validate a resource on multiple Mirror Servers.

Success: 2.01 "Created", the request MUST include the initial

representation of the mirrored resource.

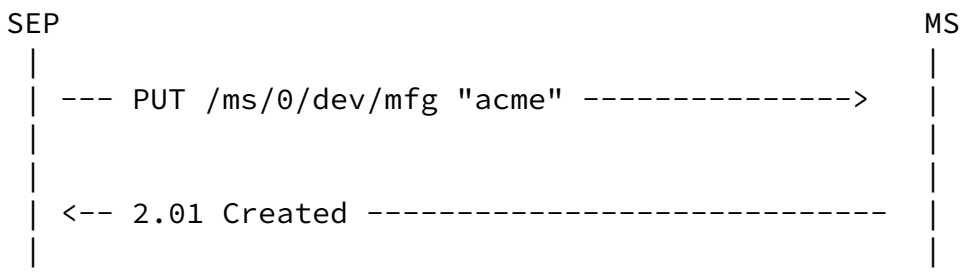
Success: 2.04 "Changed", the request MUST include the new representation of the mirrored resource.

Success: 2.05 "Content", the response MUST include the current representation of the mirrored resource.

Failure: 4.00 "Bad Request". Malformed request.

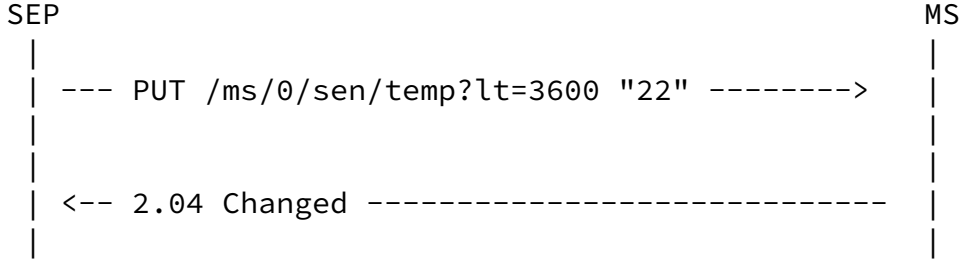
Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example describes how a sleeping endpoint can initialize the resource containing its manufacturer name just after registration.



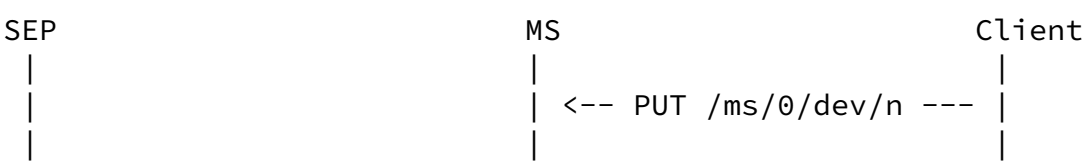
Req: PUT /ms/0/dev/mfg
Payload: acme
Res: 2.01 Created

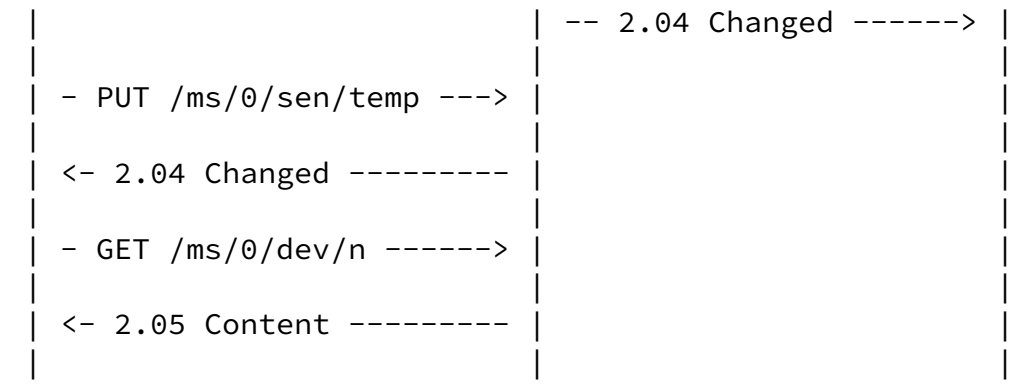
The example below shows how a SEP can indicate that it is supposed to send a temperature value at least every hour to keep its mirror entry active.



```
Req: PUT /ms/0/sen/temp?lt=3600
Payload: 22
Res: 2.04 Changed
```

The following example shows a commissioning tool changing the name of a sleeping device through a Mirror Server. The SEP detects this change right after mirroring its current temperature.





Req: PUT /ms/0/dev/n
Payload: "sensor-1"
Res: 2.04 Changed

Req: PUT /ms/0/sen/temp
Payload: "24"
Res: 2.04 Changed, Content-Type: application/link-format
Payload: "</ms/0/dev/n>"

Req: GET /ms/0/dev/n
Res: 2.05 Content
Payload: "sensor-1"

4.7. Client Operation

The Client Operation interface is not defined for a Resource Directory and is specific to the Mirror Server Function Set.

While the SEP operation interface describes only the interaction between the SEP and the MS on mirrored resources, the interface between a client and the MS for mirrored resources is actually defined by the Link Format payload at registration. This specification does not define the list of Interface Description attribute values that a Mirror Server must support. This is left to a companion specification such as a CoRE profile specification. A Mirror Server MAY support complex interfaces that require special logic and interactions between multiple mirrored resources. The CoRE Batch interface defined in [I-D.shelby-core-interfaces] is an example of complex interface that defines relations between a parent resource and sub-resources using SenML [I-D.jennings-senml].

A SEP may register resources with the "obs" attribute. In that case a MS using the CoAP protocol [I-D.ietf-core-coap] SHOULD accept to establish a CoAP observation relationship between the observable mirrored resource and a client as defined in [I-D.ietf-core-observe].

A SEP may stop updating its mirrored resources without explictly removing its mirror entry (e.g. transition to another MS after network unreachability detection). A client can detect this situation when the corresponding mirror entry has expired. Upon receipt of a response with error code 4.04 "Not Found", a client SHOULD restart resource discovery to determine if the resources are now mirrored on another MS.

The Client operation interface is specified as follows:

Interaction: Client -> MS

Method: Defined at registration

URI Template: /{+location}{+resource}

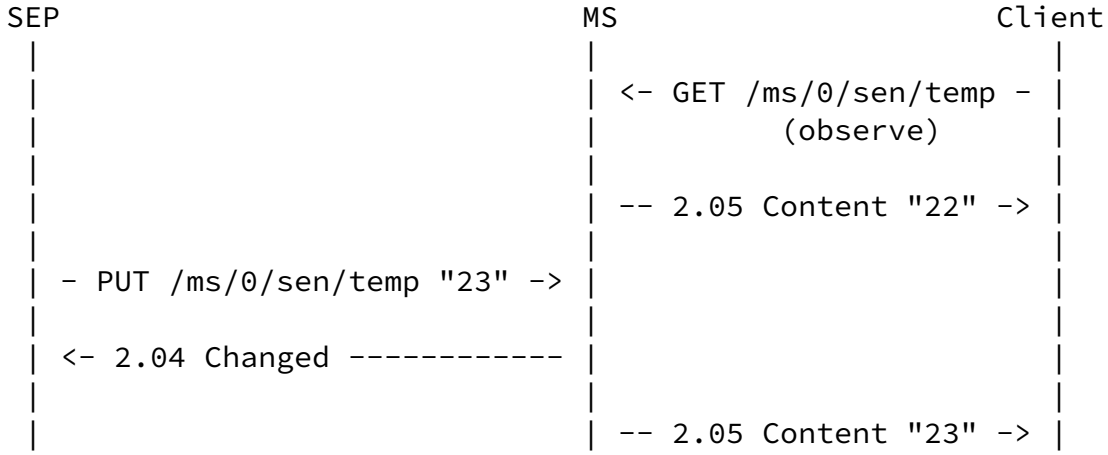
URI Template Variables:

location := This is the Location path returned by the MS as a result of a successful registration.

resource := This is the relative path to a mirrored resource managed by a SEP.

Content-Type: Defined at registration

In the example below a client observes the changes of temperature through the Mirror Server.



4.8. Modification check

This interface is not defined for a Resource Directory and is specific to the Mirror Server Function Set.

A sleeping endpoint may register resources supporting POST or PUT methods and therefore that could be modified by clients. In that case the SEP needs to poll these resources to detect updates. Polling each modifiable resource is inefficient when they are numerous. The modification check interface allows a SEP to retrieve a list of resources that have been modified. The SEP can then send GET requests on each resource of the list to get the updated representation. A POST request on the check interface automatically clears the list of modified resources.

The check interface is specified as follows:

Interaction: SEP -> MS

Method: POST

URI Template: /{+location}?chk

URI Template Variables:

location := This is the Location path returned by the MS as a result of a successful registration.

Request Content-Type: None

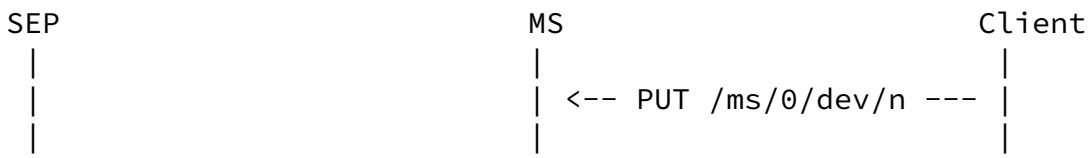
Response Content-Type: application/link-format

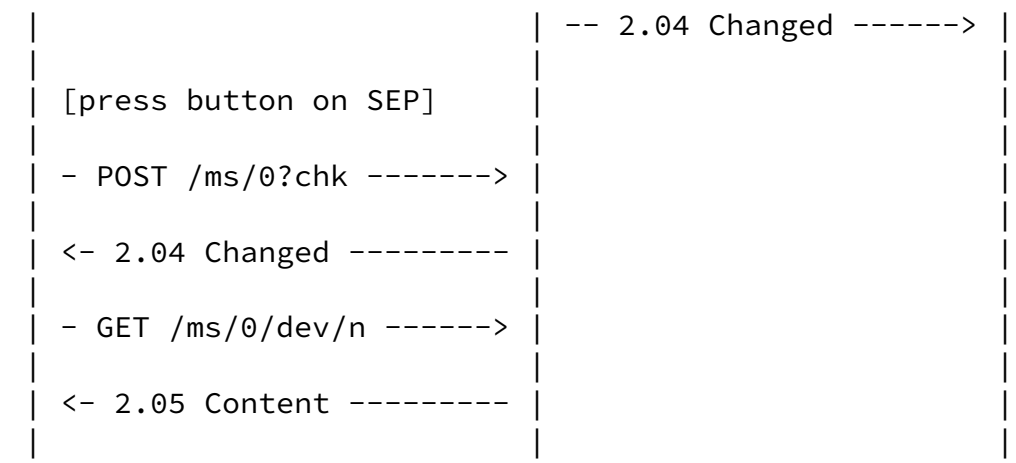
Success: 2.04 "Changed", the response MUST include a list of web links to resources that have been modified by clients.

Failure: 4.00 "Bad Request". Malformed request.

Failure: 5.03 "Service Unavailable". Service could not perform the operation.

The following example shows a commissioning tool changing the name of a sleeping device through a Mirror Server. A commissioning button on the SEP triggers the reading of the new configuration.





Req: PUT /ms/0/dev/n
Payload: "sensor-1"
Res: 2.04 Changed

Req: POST /ms/0?chk
Res: 2.04 Changed
Payload: "</ms/0/dev/n>"

Req: GET /ms/0/dev/n
Res: 2.05 Content
Payload: "sensor-1"

5. Acknowledgements

Thanks to Zach Shelby who is the author of the Resource Directory interface. Thanks to Nicolas Riou, Jari Arkko, Esko Dijk and Carsten Bormann who have provided useful comments.

6. IANA Considerations

"core.ms" resource type needs to be registered.

The "ep" attribute needs to be registered.

7. Security Considerations

This document needs the same security considerations as described in Section 7 of [RFC5988] and Section 6 of [I-D.ietf-core-link-format].

The Mirror Server architecture defines the SEP and Client roles in the Mirror Function Set interfaces. Since the roles are based on the requester identity, a MS SHOULD perform appropriate authentication in order to prevent a malicious client endpoint from impersonating the SEP or an authorized client. Otherwise the malicious client could

gain access to interfaces allowing corruption or deletion of a mirrored resource.

A malicious client could start a denial of service attack by trying to mirror a large resource on a MS. Memory exhaustion would prevent other sleeping endpoints from mirroring their resources. A MS SHOULD use quotas to limit the size and the number of mirrored resources per SEP.

A Mirror Server is actually an intermediary running at application level. As a consequence the Mirror Server architecture can only provide implicit end-to-end security that relies on a trusted network if security is not available at application layer. When explicit end-to-end security is required between a SEP and a Client, data object security SHOULD be employed.

8. Changelog

Changes from -00 to -01

- Lightweight modification check mechanism.

9. References

9.1. Normative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-14 (work in progress), March 2013.

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

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

[I-D.shelby-core-interfaces]
Shelby, Z. and M. Vial, "CoRE Interfaces", draft-shelby-core-interfaces-05 (work in progress), March 2013.

[I-D.shelby-core-resource-directory]
Shelby, Z., Krco, S., and C. Bormann, "CoRE Resource Directory", draft-shelby-core-resource-directory-05 (work in progress), February 2013.

[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.

9.2. Informative References

[I-D.arkko-core-sleepy-sensors]
Arkko, J., Rissanen, H., Loreto, S., Turanyi, Z., and O. Novo, "Implementing Tiny COAP Sensors", draft-arkko-core-sleepy-sensors-01 (work in progress), July 2011.

[I-D.jennings-senml]
Jennings, C., Shelby, Z., and J. Arkko, "Media Types for Sensor Markup Language (SENML)", draft-jennings-senml-10 (work in progress), October 2012.

[I-D.shelby-core-coap-req]
Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R.

Kelsey, "CoAP Requirements and Features", draft-shelby-core-coap-req-02 (work in progress), October 2010.

Author's Address

Matthieu Vial
Schneider-Electric
Grenoble
FRANCE

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