GWD-R
OCCI-WG

Ralf Nyrén

February 4, 2013
Updated: January 21, 2015

Open Cloud Computing Interface - RESTful HTTP Protocol

Status of this Document

This is a draft proposal of a new document in the Open Cloud Computing Interface specification suite.

Copyright Notice

Trademarks

Abstract

This document, part of a document series, produced by the OCCI working group within the Open Grid Forum (OGF), provides a high-level definition of a Protocol and API. The document is based upon previously gathered requirements and focuses on the scope of important capabilities required to support modern service offerings.

Comments

- ...

Contents

1 Introduction

2 Notational Conventions

All these parts and the information within are mandatory for implementors (unless otherwise specified). 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 [?].

The following terms [?] are used when referring to URI components:

```
 http://example.com:8080/over/there?action=stop#xyz
 \__/   \______________/\_________/ \_________/ \_/
   |           |             |            |        |
scheme     authority        path        query   fragment
```

3 OCCI RESTful HTTP Protocol

This document specifies the OCCI HTTP Protocal, a RESTful protocol for communication between OCCI Server and OCCI Client. The OCCI HTTP Protocol support multiple different data formats as payload. Data formats are specified an separate documents.

TBD: general intro to REST etc

4 Namespace

The OCCI HTTP Protocol maps the OCCI Core model into the URL hierarchy by binding Kind and Mixin instances to unique URL paths. Such a URL path is called the *location* of the Kind or Mixin. A provider is free to choose the *location* as long as it is unique within the service provider's URL namespace. For example, the Kind instance[1] for the Compute type may be bound to `/my/occi/api/compute/`.

A Kind instance whose associated type cannot be instantiated MUST NOT be bound to an URL path. This applies to the Kind instance for OCCI Entity which, according to OCCI Core, cannot be instantiated [?].

TODO: sub resources here.

4.1 Bound and unbound paths

Since a limited set of URL paths are bound to Kind and Mixin instances the URL hierarchy consists of both *bound* and *unbound* paths. A bound URL path is the *location* of a Kind or Mixin collection.

An unbound URL path MAY represent the union of all Kind and Mixin collection "below" the unbound path. TODO: FIXME: Should this be a MUST instead?

5 Headers and status codes

TODO: add all other HTTP niftyness from old spec here.

OCCI Clients and Servers must include a minimum set of mandatory HTTP headers in each request and response in order to be compliant. There is also a minimum set of HTTP status codes which must be supported by an implementation of the OCCI HTTP Protocol.

[1]`http://schemas.ogf.org/occi/infrastructure#compute`

5.1 Mandatory HTTP requests headers

Accept An OCCI Client SHOULD specify the media-types the OCCI data formats it supports in the `Accept` header.

Content-type If an OCCI Client submits payload in a HTTP request the OCCI Client MUST specify the media-type of the OCCI data format in the `Content-type` header.

5.2 Mandatory HTTP response headers

Content-type An OCCI Server MUST specify the media-type of the OCCI data format used in a HTTP Response.

Server An OCCI Server MUST specify the OCCI HTTP Protocol version number.

5.3 HTTP status codes

The below list specifies the minimum set of HTTP status codes an OCCI Client MUST understand. An OCCI Server MAY return other HTTP status codes but the exact client behaviour in such cases is not specified.

200

204

301

400

403

404

405

6 HTTP methods applied to entity instance URLs

TODO: Sanity check this section against v1.1...

This section describes the HTTP methods used to retrieve and manipulate individual entity instances. An *entity instance* refers to an instance of the OCCI Resource type, OCCI Link type or a sub-type thereof [?].

Each HTTP method described is assumed to operate on an URL referring to a single element in a collection, an URL such as the following:

```
  http://example.com/compute/012d2b48-c334-47f2-9368-557e75249042
```

6.1 GET entity instance

The HTTP GET method retrieves a representation of a single (existing) entity instance.

6.1.1 Client GET request

The body of the HTTP GET request MUST be empty.

```
GET /compute/012d2b48-c334-47f2-9368-557e75249042 HTTP/1.1
Host: example.com
Accept: application/occi+xxx
User-Agent: occi-client/x.x OCCI/1.1
```

6.1.2 Server GET response

The body of the HTTP GET response MUST contain a representation of the entity instance.

```
HTTP/1.1 200 OK
Server: occi-server/x.x OCCI/1.1
Content-Type: application/occi+xxx; charset=utf-8

...
```

6.2 PUT entity instance

The HTTP PUT method either *creates* a new or *replaces* an existing entity instance at the specified URL. The unique identifier of the entity instance (Entity.id) MUST be specified in the request URL. An OCCI Client MAY also specify Entity.id in the payload however it MUST be identical to the identifier specified as part of the request URL.

If an entity instance with the specified identifier does not exist the PUT request allows the OCCI Client to choose the Entity.id of the new instance. An OCCI Server MAY refuse such a request with HTTP status code 405. This would indicate that the OCCI Server does not allow user defined entity identifiers.

The PUT method MUST be idempotent, i.e. multiple identical PUT requests should have the same effect as a single request.

6.2.1 Client PUT request

The full representation of the entity instance MUST be supplied in the HTTP body of the request. The request body MUST only include a representation of a single entity instance.

If the request represent an OCCI Resource (as opposed to an OCCI Link) the representation MUST NOT include any Link instances associated with the Resource instance. A server MUST refuse a request including associated Links

Any OCCI Links associated with an existing OCCI Resource MUST be left intact.

```
PUT /compute/012d2b48-c334-47f2-9368-557e75249042 HTTP/1.1
Host: example.com
Accept: application/occi-entity+json
User-Agent: occi-client/x.x OCCI/1.1
Content-Type: application/occi-entity+json; charset=utf-8

{
  "kind": "http://schemas.ogf.org/occi/infrastructure#compute",
  "mixins": [ ... ],
  "attributes": { ... }
}
```

6.2.2 Server PUT response

Content-Type `application/occi-entity+json`

Upon success an OCCI server MUST return HTTP status code 200 and a complete JSON representation of the created/replaced entity instance in single-entity-instance format. The response MUST be identical[2] to that of a subsequent GET request of the same URL.

[2]Provided the entity instance was not changed in the meantime.

```
HTTP/1.1 200 OK
Server: occi-server/x.x OCCI/1.1
Content-Type: application/occi-entity+json; charset=utf-8

{
  "kind": "http://schemas.ogf.org/occi/infrastructure#compute",
  "mixins": [ ... ],
  "actions": [ ... ],
  "links": [ ... ],
  "attributes": { ... }
}
```

6.3 POST entity instance (action)

There are two methods to invoke an OCCI Action using the JSON Rendering.

1. Supply the query parameter "action" together with the request. The value of "action" MUST be the `term` of the action Category.
2. Specify `application/occi-action+json` in the Content-Type header and supply a request payload formatted according to section **??**. In order to specify action attributes this method MUST be used.

An OCCI Client MAY combine the two methods if the "action" parameter's value is equal to the Category `term` in the body.

6.3.1 Client POST action request

Accept `application/occi-entity+json`

Content-Type `application/occi-action+json`

The example shows the combined method.

```
POST /compute/012d2b48-c334-47f2-9368-557e75249042?action=stop HTTP/1.1
Host: example.com
Accept: application/occi-entity+json
User-Agent: occi-client/x.x OCCI/1.1
Content-Type: application/occi-action+json; charset=utf-8

{
  "category": "http://schemas.ogf.org/occi/infrastructure/compute/action#stop",
  "attributes": {
    "method": "graceful"
  }
}
```

6.3.2 Server POST action response

Content-Type `application/occi-entity+json`

If the request Accept header contains `application/occi-entity+json` the server MAY return status code 200 and a full representation of the entity instance. Otherwise the server MUST return status code 204 and no response payload.

```
HTTP/1.1 204 OK
Server: occi-server/x.x OCCI/1.1
```

6.4 POST entity instance

Content-Type `application/occi-entity+json`

TODO: This would imply a partial update of the entity instance. While it is easy to supply only the attributes to be updated the question is if there are any valid use cases for partial updates using JSON?

6.5 DELETE entity instance

The HTTP DELETE method destroys an entity instance and any OCCI Links associated with an OCCI Resource.

6.5.1 Client DELETE request

Content-Type `application/occi-entity+json`

```
DELETE /compute/012d2b48-c334-47f2-9368-557e75249042 HTTP/1.1
Host: example.com
Accept: application/occi-entity+json
```

6.5.2 Server DELETE response

```
HTTP/1.1 204 OK
Server: occi-server/x.x OCCI/1.1
```

7 HTTP methods applied to collections URLs

TODO: NOT fully updated yet!

This section describes the HTTP methods used to manipulate collections. Each HTTP method described is assumed to operate on an URL referring to a collection of elements, an URL such as the following:

```
  http://example.com/storage/
```

A collection consist of a set of entity instances and there are three different types of collections which may be exposed by an OCCI server. The request and response format is identical for all three types collections although the semantics differ slightly for the PUT and POST methods.

Kind locations The location associated with an OCCI Kind instance represents the collection of all entity instances of that particular Kind.

Mixin locations The location of an OCCI Mixin instance represents the collection of all entity instances associated with that Mixin.

Arbitrary path Any path in the URL namespace which is neither a Kind nor a Mixin location. A typical example is the root URL e.g. `http://example.com/`. Such a path combines all collections in the sub-tree starting at the path. Therefore the root URL is a collection of all entity instances available.

7.1 GET collection

The HTTP GET method retrieves a list of all entity instances in the collection. Filtering and pagination information is encoded in the query string of the URL.

7.1.1 Client GET request

The query string of the request URL MUST have the following format:

```
query-string        = ""
                    | "?" query-parameter *( "&" query-parameter )
  query-parameter   = attribute-filter
                    | category-filter
                    | pagination-marker
                    | pagination-limit
  attribute-filter  = "q=" attribute-search *( "+" attribute-search )
  attribute-search  = 1*( string-urlencoded |
                          attribute-name "%3D" string-urlencoded )
  category-filter   = "category=" string-urlencoded
  pagination-marker = "marker=" UUID
  pagination-limit  = "limit=" 1*( DIGIT )
  attribute-name    = attr-component *( "." attr-component )
  attr-component    = LOALPHA *( LOALPHA | DIGIT | "-" | "_" )
  string-urlencoded = *( ALPHA | DIGIT | "-" | "_" | "." | "~" | "%" )
```

TODO: FIXME: UUID in ABNF

Filtering A search filter can be applied to categories and attributes of entity instances in a collection. An OCCI server SHOULD support filtering. The query parameters MUST be URL encoded.

Attribute filters are specified using the *q* query parameter. A filter such as q=ubuntu+inactive would match all entity instances whose combined set of attribute values includes both the word "ubuntu" and "inactive". It is also possible to match on specific attributes by preceding the search term with the attribute name and an equal sign, for example `occi.core.title%3Dubuntu+occi.compute.state%3Dinactive`.

The category filter is specified using the *category* query parameter and represent a single Kind, Mixin or Action category to be matched. The following query would include only entity instances of the Compute type: `category=http%3A%2F%2Fschemas.ogf.org%2Focci%2Finfrastructure%23compute`

Pagination TODO: FIXME: marker instead of start An OCCI client MAY request that the server only return a subset of a collection. This is accomplished using the *marker* and *limit* query parameters. An OCCI server MUST support pagination.

The *marker* parameter specifies the offset into the collection. A value of zero, `marker=0` indicates the beginning of the collection. The *limit* parameter sets the maximum number of elements to include in the response. For example `?marker=...&limit=10` would indicate the third page with a limit of 10 elements per page.

Example request

```
GET /storage/?q=ubuntu+server&limit=20 HTTP/1.1
Host: example.com
Accept: application/occi-collection+json
User-Agent: occi-client/x.x OCCI/1.1
```

7.1.2 Server GET response

```
HTTP/1.1 200 OK
Server: occi-server/x.x OCCI/1.1
Content-Type: application/occi-collection+json; charset=utf-8
```

```

{
  "collection": [
    {
      "kind": "..." ,
      "mixins": [ ... ],
      "actions": [ ... ],
      "links": [ ... ],
      "attributes": { ... },
    },
    { ... },
    { ... }
  ],
  "limit": 20,
  "size": 137,
  "next": "http://example.com/storage/?q=ubuntu+server&marker=59b50...9b3&limit=20"
}
```

7.2 POST collection

The HTTP POST method is used to create/update one or more entity instances in a single atomic request. An OCCI server MUST identify existing entity instances using the `occi.core.id` attribute.

7.2.1 Client POST request

```
POST /storage/ HTTP/1.1
Host: example.com
Accept: application/occi-collection+json
User-Agent: occi-client/x.x OCCI/1.1
Content-Type: application/occi-collection+json; charset=utf-8

{
  "collection": [
    {
      "kind": "...",
      "mixins": [ ... ],
      "links": [ ... ],
      "attributes": { ... },
    },
    { ... },
    { ... }
  ]
}
```

7.2.2 Server POST response

```
HTTP/1.1 204 OK
Server: occi-server/x.x OCCI/1.1
```

TODO: Should we support HTTP 200 returning the whole collection? Or maybe just the entity instances created/updated?

7.3 POST collection with “action” query parameter

todo

7.4 PUT collection

Replace the entire collection with a new one. TODO: Should we support this?

7.5 DELETE collection

Delete the entire collection. TODO: Should we support this?

8 HTTP methods applied to QI

TODO: write this.

9 Glossary

Term	Description
Action	An OCCI base type. Represents an invocable operation on a Entity sub-type instance or collection thereof.
Attribute	A type in the OCCI Core Model. Describes the name and properties of attributes found in Entity types.
Category	A type in the OCCI Core Model and the basis of the OCCI type identification mechanism. The parent type of Kind.
capabilities	In the context of Entity sub-types **capabilities** refer to the OCCI Attributes and OCCI Actions exposed by an **entity instance**.
Client	An OCCI client.
Collection	A set of Entity sub-type instances all associated to a particular Kind or Mixin instance.
Entity	An OCCI base type. The parent type of Resource and Link.
entity instance	An instance of a sub-type of Entity but not an instance of the Entity type itself. The OCCI model defines two sub-types of Entity, the Resource type and the Link type. However, the term *entity instance* is defined to include any instance of a sub-type of Resource or Link as well.
Kind	A type in the OCCI Core Model. A core component of the OCCI classification system.
Link	An OCCI base type. A Link instance associates one Resource instance with another.
Mixin	A type in the OCCI Core Model. A core component of the OCCI classification system.
mix-in	An instance of the Mixin type associated with an *entity instance*. The “mix-in” concept as used by OCCI *only* applies to instances, never to Entity types.
model attribute	An internal attribute of a the Core Model which is *not* client discoverable.
OCCI	Open Cloud Computing Interface.
OCCI base type	One of Entity, Resource, Link or Action.
OCCI Action	see Action.
OCCI Attribute	A client discoverable attribute identified by an instance of the Attribute type. Examples are occi.core.title and occi.core.summary.
OCCI Category	see Category.
OCCI Entity	see Entity.
OCCI Kind	see Kind.
OCCI Link	see Link.
OCCI Mixin	see Mixin.
OGF	Open Grid Forum.
Resource	An OCCI base type. The parent type for all domain-specific Resource sub-types.
resource instance	See *entity instance*. This term is considered obsolete.
tag	A Mixin instance with no attributes or actions defined.
template	A Mixin instance which if associated at instance creation-time pre-populate certain attributes.
type	One of the types defined by the OCCI Core Model. The Core Model types are Category, Attribute, Kind, Mixin, Action, Entity, Resource and Link.
concrete type/sub-type	A concrete type/sub-type is a type that can be instantiated.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
URN	Uniform Resource Name.

10 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the

11 Disclaimer

12 Full Copyright Notice