Draft
OCCI-WG

Ralf Nyrén, Independent
Andy Edmonds, ICCLab, ZHAW
Thijs Metsch, Intel
Boris Parák, CESNET

Updated: March 19, 2015

Open Cloud Computing Interface - HTTP Protocol

Status of this Document

This document is a draft providing information to the community regarding the specification of the Open Cloud Computing Interface.

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.

Contents

1 Introduction

The Open Cloud Computing Interface (OCCI) is a RESTful Protocol and API for all kinds of management tasks. OCCI was originally initiated to create a remote management API for IaaS[1] model-based services, allowing for the development of interoperable tools for common tasks including deployment, autonomic scaling and monitoring. It has since evolved into a flexible API with a strong focus on interoperability while still offering a high degree of extensibility. The current release of the Open Cloud Computing Interface is suitable to serve many other models in addition to IaaS, including PaaS and SaaS.

In order to be modular and extensible the current OCCI specification is released as a suite of complimentary documents, which together form the complete specification. The documents are divided into four categories consisting of the OCCI Core, the OCCI Protocols, the OCCI Renderings and the OCCI Extensions.

- The OCCI Core specification consists of a single document defining the OCCI Core Model. The OCCI Core Model can be interacted through *renderings* (including associated behaviours) and expanded through *extensions*.
- The OCCI Protocol specifications consist of multiple documents each describing how the model can be interacted with over a particular protocol (e.g. HTTP, AMQP etc.). Multiple protocols can interact with the same instance of the OCCI Core Model.
- The OCCI Rendering specifications consist of multiple documents each describing a particular rendering of the OCCI Core Model. Multiple renderings can interact with the same instance of the OCCI Core Model and will automatically support any additions to the model which follow the extension rules defined in OCCI Core.
- The OCCI Extension specifications consist of multiple documents each describing a particular extension of the OCCI Core Model. The extension documents describe additions to the OCCI Core Model defined within the OCCI specification suite.

The current specification consists of seven documents. This specification describes version 1.2 of OCCI and is backward compatible with 1.1. Future releases of OCCI may include additional protocol, rendering and extension specifications. The specifications to be implemented (MUST, SHOULD, MAY) are detailed in the table below.

Table 1. What OCCI specifications must be implemented for the specific version.

Document	OCCI 1.1	OCCI 1.2
Core Model	MUST	MUST
Infrastructure Model	SHOULD	SHOULD
Platform Model	MAY	MAY
SLA Model	MAY	MAY
HTTP Protocol	MUST	MUST
Text Rendering	MUST	MUST
JSON Rendering	MAY	MUST

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 [1].

The following terms [2] are used when referring to URL components:

[1] Infrastructure as a Service

```
 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 Protocol, 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.

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[2] for the Compute type may be bound to `/my/occi/api/compute/`.

Whenever a *location* is rendered it MUST be either a String or as defined in RFC6570 [3].

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 [4].

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.

5 Headers and Status Codes

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.

5.1 Requests Headers

Accept An OCCI client SHOULD specify the media types of 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.

User-Agent An OCCI client SHOULD specify the OCCI version number in the `User-Agent` header. See Section 5.3.

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

5.2 Response Headers

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

Content-type An OCCI server MUST specify the media type of the OCCI data format used in an HTTP response.

Server An OCCI server MUST specify the OCCI version number in the `Server` header. See Section 5.3.

5.3 Versioning

Information about the OCCI version supported by a server implementation MUST be advertised to a client on each response. The version field in the response MUST include the value OCCI/X.Y, where X is the major version number and Y is the minor version number of the implemented OCCI version. The server response MUST relay versioning information using the HTTP ‘Server’ header.

```
HTTP/1.1 200 OK
Server: occi-server/1.1 (linux) OCCI/1.2
[...]
```

Complementing the server-side behavior of an OCCI implementation, a client SHOULD indicate the version it expects to interact with. In a client, this information SHOULD be advertised in all requests it issues. A client request SHOULD relay versioning information in the ‘User-Agent’ header. The ‘User-Agent’ header MUST include the same value (OCCI/X.Y) as advertised by the server.

```
GET /-/ HTTP/1.1
Host: example.com
User-Agent: occi-client/1.1 (linux) libcurl/7.19.4 OCCI/1.2
[...]
```

If an OCCI implementation receives a request from a client that supplies a version number higher than the server supports, the server MUST respond back to the client with an HTTP status code indicating that the requested version is not implemented. The *HTTP 501 Not Implemented* status code MUST be used.

OCCI implementations compliant with this version of the document MUST use the version string *OCCI/1.2*. Versioning of extensions is out of scope for this document.

5.4 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 behavior in such cases is not specified. The return codes are specified by [5] and [6].

200 OK indicates that the request has succeeded.

201 CREATED indicates that the request has been fulfilled and has resulted in one or more new resources being created.

400 Bad Request indicates that the server cannot or will not process the request due to something that is perceived to be a client error

401 Unauthorized indicates that the request has not been applied because it lacks valid authentication credentials for the target resource.

403 Forbidden indicates that the server understood the request but refuses to authorize it.

404 Not Found indicates that the origin server did not find a current representation for the target resource or is not willing to disclose that one exists

405 Method Not Allowed indicates that the method received in the request-line is known by the origin server but not supported by the target resource.

406 Not Acceptable indicates that the target resource does not have a current representation that would be acceptable to the user agent

409 Conflict indicates that the request could not be completed due to a conflict with the current state of the resource

413 Request Entity Too Large indicates that the request is larger than the server is willing or able to process.

500 Internal Server Error indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

501 Not Implemented indicates that the server does not support the functionality required to fulfill the request.

503 Service Unavailable indicates that the server is currently unable to handle the request due to a temporary overload or maintenance of the server

6 Pagination

To request partial results of an otherwise large collection message response, pagination SHOULD be used to reduce the load on both the client and the service provider. This is done in the following manner.

The HTTP GET verb is used when accessing a URL of a collection and the query parameters of *page* and *number* MUST be used. *page* is an indexed integer that refers to a sub-collection of the requested collection. *number* is an integer of items that SHOULD be displayed in one paged response.

If *number* is too large for the provider to handle (policy, technical limitations) then an *HTTP 413 Request Entity Too Large* response status code MUST be issued to the requesting client.

If there is no more content to be served, the response status code issued to the requesting client MUST be an *HTTP 200 OK* and the response body MUST contain an empty collection.

7 HTTP Methods Applied to Query Interface

This section describes the HTTP methods used to retrieve and manipulate category instances. With the help of the query interface it is possible for the client to determine the capabilities of the OCCI implementation he refers to.

The query interface MUST be implemented by all OCCI implementations. It MUST be found at:

```
/-/
```

Implementations MAY also adopt RFC5785 [7] compliance to advertise this location. Should implementations wish to advertise the Query Interface using the .well-known mechanism then they MUST use the following path served from the authority:

```
/.well-known/org/ogf/occi/-/
```

The renderings for the *category* instance and category *collection* are defined in [8] and [9].

7.1 GET Method

Client GET request

The request MAY include a possible filter rendering.

Server GET response

The response MUST include a category collection rendering.

Upon a successfully request a *200 OK* status code MUST be used.

7.2 PUT Method

N/A

7.3 POST Method

Client POST request

The request MUST include at least one full category instance rendering. It MAY include a category collection rendering.

Server POST response

Upon a successful processing of the request, the *200 OK* status code MUST be returned.

7.4 DELETE Method

Client DELETE request

The request MUST include at least one full category instance rendering. It MAY include a category collection rendering.

Server DELETE response

Upon a successful processing of the request, the *200 OK* status code MUST be returned.

8 HTTP Methods Applied to Entity Instance

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 [4].

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
```

The renderings for the *entity* and *action* instances are defined in [8] and [9].

8.1 GET Method

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

Client GET request

N/A

Server GET response

The response MUST contain an entity instance rendering.

Upon a successful processing of the request, the *200 OK* status code MUST be returned.

8.2 PUT Method

The HTTP PUT method either *creates* a new or *replaces* an existing entity instance at the specified URL.

8.2.1 Create

Client PUT request
The request MUST contain an entity instance rendering.

Server PUT response
The OCCI implementation MAY return either the *201 Created* or *200 OK* status code. If the OCCI implementation returns the *200 OK* status code, an entity instance rendering MUST be included as well. In case of the *201 Created* status code, a location (as defined in RFC7231 [5]) MUST be included.

8.2.2 Replace

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

Client PUT request
The request MUST contain an entity instance rendering.

Server PUT response
The OCCI implementation MAY return either the *201 Created* or *200 OK* status code. If the OCCI implementation returns the *200 OK* status code, an entity instance rendering MUST be included as well. In case of the *201 Created* status code, a location (as defined in RFC7231 [5]) MUST be included.

8.3 POST Method

The HTTP POST method either *partially update*s an existing entity instance or triggers an *action* on an existing entity instance.

8.3.1 Partial Update

Client POST request
The request MUST contain a partial entity instance rendering of the entity instance to be changed.

Server POST response
The OCCI implementation MAY return either the *201 Created* or *200 OK* status code. If the OCCI implementation returns the *200 OK* status code, an entity instance rendering MUST be included as well. In case of the *201 Created* status code, a location (as defined in RFC7231 [5]) MUST be included.

8.3.2 Trigger Action

Actions are triggered using the HTTP POST verb and by adding a query string to the URL. This query MUST contain a key-value pair. The key MUST be ‘action’. The value MUST equal to the Action’s term.

Client POST request
The request MUST contain an action invocation rendering.

Server POST response
The response of the HTTP GET response MUST contain an entity instance rendering.

Upon a successful processing of the request, the *200 OK* status code MUST be returned.

8.4 DELETE Method

The HTTP DELETE method deletes an entity instance

Client DELETE request

N/A

Server DELETE response

Upon a successful processing of the request, the *200 OK* status code MUST be returned.

9 HTTP Methods Applied to Collection

This section describes the HTTP methods used to retrieve and manipulate collections. A collection refers to a set of *entity instance*s.

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

```
http://example.com/compute/
```

The renderings for the entity instance, entity *collection* and *action* instances are defined in [8] and [9].

9.1 GET Method

The HTTP GET method retrieves a rendering of a collection of existing entity instances.

Client GET request

The request MAY include a possible filter rendering.

Server GET response

The response MUST include an entity collection rendering.

Upon a successful processing of the request, the *200 OK* status code MUST be returned.

9.2 PUT Method

The HTTP PUT is only defined for a collection defined by a Mixin. It makes replacing the collection possible.

Client PUT request

The request MUST include an entity collection rendering.

Server PUT response

The response MUST include an entity collection rendering.

Upon a successful processing of the request, the *200 OK* status code MUST be returned.

9.3 POST Method

The HTTP POST method is defined for *creation* of an entity instance, *association* of entity instance with a Mixin and triggering *action*s.

9.3.1 Create Entity Instance

Client POST request
The request MUST include at least one full entity instance rendering. It MAY include an entity collection rendering.

Server POST response
The OCCI implementation MAY return either the *201 Created* or *200 OK* status code. If the OCCI implementation returns the *200 OK* status code, an entity instance rendering or collection rendering MUST be included as well. In case of the *201 Created* status code, an entity instance location (as defined in RFC7231 [5]) or a list of entity instance locations MUST be included.

9.3.2 Associate Mixin with Entity Instance

This operation MUST only be available for collections defined by a Mixin.

Client POST request
The request MUST include an entity collection rendering which require the Mixin to be applied.

Server POST response
On successful operation the server replies with the *200 OK* HTTP status code it MUST include an entity collection rendering.

9.3.3 Trigger Action

Actions are triggered using the HTTP POST verb and by adding a query string to the URL. This query MUST contain a key-value pair. The key MUST be ‘action’. The value MUST equal to the Action’s term.

Client POST request
The request MUST contain an action invocation rendering.

Server POST response
The response of the HTTP GET response MUST contain an entity collection rendering.

Upon a successful processing of the request, the *200 OK* status code MUST be returned.

9.4 DELETE Method

The HTTP delete method is used to either *delete* all entity instances in a collection or *disassociate* entity instance from a collection defined by a Mixin.

9.4.1 Delete Entity Instances

Client DELETE request
N/A

Server DELETE response
Upon a successful processing of the request, the *200 OK* status code MUST be returned.

9.4.2 Disassociate Mixin from Entity Instances

This operation MUST only be available for collections defined by a Mixin.

Client DELETE request
The request MAY include entity collection rendering which requires the Mixin to be disassociated.

Server DELETE response
Upon a successful processing of the request, the *200 OK* status code MUST be returned.

10 Security Considerations

The OCCI HTTP rendering assumes HTTP or HTTP-related mechanisms for security. As such, implementations SHOULD support TLS [3] for transport layer security.

Authentication SHOULD be realized by HTTP authentication mechanisms, namely HTTP Basic or Digest Auth [10], with the former as default. Additional profiles MAY specify other methods and should ensure that the selected authentication scheme can be rendered over the HTTP or HTTP-related protocols.

Authorization is not enforced on the protocol level, but SHOULD be performed by the implementation. For the authorization decision, the authentication information as provided by the mechanisms described above MUST be used.

Protection against potential Denial-of-Service scenarios is out of scope of this document; the OCCI HTTP Protocol specification assumes cooperative clients that SHOULD use selection and filtering as provided by the Category mechanism wherever possible. Additional profiles to this document, however, MAY specifically address such scenarios; in that case, best practices from the HTTP ecosystem and appropriate mechanisms as part of the HTTP protocol specification SHOULD be preferred.

As long as specific extensions of the OCCI Core and Model specification do not impose additional security requirements than the OCCI Core and Model specification itself, the security considerations documented above apply to all (existing and future) extensions. Otherwise, an additional profile to this specification MUST be provided; this profile MUST express all additional security considerations using HTTP mechanisms.

[3] http://datatracker.ietf.org/wg/tls/

11 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 Attributes and Actions exposed by an **entity instance**.
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.
OCCI	Open Cloud Computing Interface.
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. Used for taxonomic organisation of entity instances
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.

12 Contributors

We would like to thank the following people who contributed to this document:

Name	Affiliation	Contact
Michael Behrens	R2AD	behrens.cloud at r2ad.com
Mark Carlson	Toshiba	mark at carlson.net
Augusto Ciuffoletti	University of Pisa	augusto.ciuffoletti at gmail.com
Andy Edmonds	ICCLab, ZHAW	edmo at zhaw.ch
Sam Johnston	Google	samj at samj.net
Gary Mazzaferro	Independent	garymazzaferro at gmail.com
Thijs Metsch	Intel	thijs.metsch at intel.com
Ralf Nyrén	Independent	ralf at nyren.net
Alexander Papaspyrou	Adesso	alexander at papaspyrou.name
Boris Parák	CESNET	parak at cesnet.cz
Alexis Richardson	Weaveworks	alexis.richardson at gmail.com
Shlomo Swidler	Orchestratus	shlomo.swidler at orchestratus.com
Florian Feldhaus	NetApp	florian.feldhaus at gmail.com

Next to these individual contributions we value the contributions from the OCCI working group.

13 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 extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

14 Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

15 Full Copyright Notice

References

[1] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119 (Best Current Practice), Internet Engineering Task Force, Mar. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2119.txt

[2] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986 (Standard), Internet Engineering Task Force, Jan. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc3986.txt

[3] J. Gregorio, R. Fielding, M. Hadley, M. Nottingham, and D. Orchard, "URI Template," RFC 6570, Internet Engineering Task Force, Mar. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6570.txt

[4] R. Nyrén, A. Edmonds, A. Papaspyrou, and T. Metsch, "Open Cloud Computing Interface – Core," GFD-P-R.183, April 2011. [Online]. Available: http://ogf.org/documents/GFD.183.pdf

[5] R. Fielding and J. Gettys, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC 7231, Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7231.txt

[6] R. Fielding, Y. Lafon, and J. Gettys, "Hypertext Transfer Protocol (HTTP/1.1): Authentication," RFC 7235, Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7235.txt

[7] M. Nottingham, "Defining Well-Known Uniform Resource Identifiers (URIs)," RFC 5785 (Proposed Standard), Internet Engineering Task Force, Apr. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5785.txt

[8] T. Metsch and A. Edmonds, "Open Cloud Computing Interface – Text Rendering," Draft, March 2015. [Online]. Available: TBD

[9] R. Nyren and F. Feldhaus, "Open Cloud Computing Interface – JSON Rendering," Draft, March 2015. [Online]. Available: TBD

[10] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, E. Sink, and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," RFC 2617 (Standard), Internet Engineering Task Force, 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2617.txt