

GWD-R
OCCI-WG

Thijs Metsch, Platform Computing
Andy Edmonds, Intel
October 7, 2010
Updated: January 4, 2011

Open Cloud Computing Interface - RESTful HTTP Rendering

Status of this Document

This document provides information to the community regarding the specification of the Open Cloud Computing Interface. Distribution is unlimited.

Copyright Notice

Copyright ©Open Grid Forum (2009-2010). All Rights Reserved.

Trademarks

OCCI is a trademark of the Open Grid Forum.

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	3
2	Notational Conventions	3
3	A RESTful HTTP Rendering for OCCI	4
3.1	Behaviour of the HTTP Verbs	5
3.2	A RESTful Rendering of OCCI	5
3.2.1	Name-space Hierarchy and Location	5
3.2.2	Various Operations and their Prerequisites and Behaviors	7
3.3	Syntax and Semantics of the Rendering	11
3.3.1	Rendering of an OCCI-Category	11
3.3.2	Rendering of OCCI-Links and OCCI-Actions	11
3.3.3	Rendering of OCCI-Attributes	12
3.3.4	Rendering of Location-URLs	12
3.4	General HTTP Behaviors Adopted by OCCI	12
3.4.1	Security and Authentication	12
3.4.2	Caching Headers	12
3.4.3	Versioning	13
3.4.4	Content-type and Accept headers	13
3.4.5	Return codes	14
3.5	More complete examples	15
3.5.1	Creating a resource instance	15
3.5.2	Retrieving a resource instance	15
4	Contributors	16
5	Glossary	17
6	Intellectual Property Statement	17
7	Disclaimer	17
8	Full Copyright Notice	18
9	References	18

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 IaaS¹ 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 e.g. 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 three categories consisting of the OCCI Core, 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 with *renderings* (including associated behaviours) and expanded through *extensions*.
- 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 three documents. Future releases of OCCI may include additional rendering and extension specifications. The documents of the current OCCI specification suite are:

OCCI Core describes the formal definition of the the OCCI Core Model [1].

OCCI HTTP Rendering defines how to interact with the OCCI Core Model using the RESTful OCCI API [2]. The document defines how the OCCI Core Model can be communicated and thus serialised using the HTTP protocol.

OCCI Infrastructure contains the definition of the OCCI Infrastructure extension for the IaaS domain [3]. The document defines additional resource types, their attributes and the actions that can be taken on each resource type.

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 [4].

All examples in this document use one of the following three HTTP category definitions. An example namespace hierarchy is also given. Syntax and Semantics is explained in the remaining sections of the document. These examples do not strive to be complete but to show the functionalities OCCI has:

```
Category: compute;  
         class=kind;  
         scheme=http://schemas.ogf.org/occi/infrastructure#;  
         location=/compute
```

¹Infrastructure as a Service

3.1 Behaviour of the HTTP Verbs

As OCCI adopts a ROA, REST-based architecture and uses HTTP as the foundation protocol the means of interaction with all RESTful resource instances is through the four main HTTP verbs. OCCI service implementations MUST, at a minimum, support these verbs as shown in Table 1:

Table 1. HTTP Verb Behavior

Type	GET	POST (create)	POST (action)	PUT (create)	PUT (update)	DELETE
resource instance	Rendering of this resource instance	Create a new resource instance	Trigger action	Create a resource instance at the given path	Update a resource instance at an given path	Delete this resource instance
Path in the name-space hierarchy ending with /	Listing of all resource instances below this name-space	Create a new resource instance	N/A	N/A	N/A	Delete all the resource instances below this name-space hierarchy
Location of an Mixin or Kind	Listing containing locations to all resource instances belonging to this Mixin or Kind	N/A	Trigger action (defined for this kind or mixin) on all resource instances belonging to this Mixin or Kind	Add a resource instance to a Mixin	N/A	Remove a resource instance given in the request from a Mixin
Query Interface	Listing of all registered Kinds and Mixins	N/A	N/A	Add a user defined Mixin	N/A	Remove a user-defined Mixin (defined in the request)

3.2 A RESTful Rendering of OCCI

The following sections and paragraphs describe how the OCCI model MUST be implemented by OCCI implementations. Operations which are not defined are out of scope for this specification and MAY be implemented. This is the minimal set to ensure interoperability.

3.2.1 Name-space Hierarchy and Location

The name-space and the hierarchy are free definable by the Service Provider. The OCCI implementation needs to implement the location path feature, which is required by OCCI for discovering capabilities and operations on Mixins and Kinds. Location paths tell the client where all resource instance of one Kind or Mixin can be found regardless of the hierarchy the service provider defines. These paths are discoverable by the client through the Query interface 3.2.2.2.

These location paths can be part of the name-space or rendered alongside. The following example shows how the locations paths are rendered alongside the name-space hierarchy. This means that the locations paths defined by Kinds or Mixins are not part of the hierarchy of the REST Resources.

```
Category: compute;
class=kind;
scheme=http://schemas.ogf.org/occi/infrastructure#;
location=/compute
```

(This is an compute kind)

```
Category: storage;
  class=kind;
  scheme=http://schemas.ogf.org/occi/infrastructure#;
  location=/storage
```

(This is an storage kind)

```
Category: my_stuff;
  class=mixin;
  scheme=http://example.com/occi/my_stuff#;
  location=/my_stuff
```

(This is a mixin of user1)

The following namespace hierarchy is used in the examples:

```
http://example.com/-/
http://example.com/vms/user1/vm1
http://example.com/vms/user1/vm2
http://example.com/vms/user2/vm1
http://example.com/disks/user1/disk1
http://example.com/disks/user2/disk1
http://example.com/compute/
http://example.com/storage/
http://example.com/my_stuff/
```

Location paths can also be part of the name-space hierarchy. In this case the location is embedded in the hierarchy of resources (e.g. the location of the compute Kind is part of <http://example.com/vms/user1/vm1>).

```
Category: compute;
  class=kind;
  scheme=http://schemas.ogf.org/occi/infrastructure#;
  location=/vms
```

(This is an compute kind)

```
Category: storage;
  class=kind;
  scheme=http://schemas.ogf.org/occi/infrastructure#;
  location=/disks
```

(This is an storage kind)

```
Category: my_stuff;
  class=mixin;
  scheme=http://example.com/occi/my_stuff#;
  location=/my_stuff
```

(This is a mixin of user1)

The following namespace hierarchy is used in the examples:

```
http://example.com/-/
http://example.com/vms/user1/vm1
http://example.com/vms/user1/vm2
http://example.com/vms/user2/vm1
http://example.com/disks/user1/disk1
http://example.com/disks/user2/disk1
http://example.com/my_stuff/
```

The flat name-space defined by OCCI where each resource instance has an identifier can be mapped to this hierarchical name-space by using the complete URIs of an resource instance as the identifier.

3.2.2 Various Operations and their Prerequisites and Behaviors

3.2.2.1 Operations on Resource Instances The following operations **MUST** be implemented by the OCCI implementation for operations on resource instances. The resource instance is uniquely identified by an URI (For example: *http://example.com/vms/user1/vm1*).²

Creating a resource instance A request to create a resource instance **MUST** contain at least one HTTP category rendering which is (or relates to) a Kind definition. If multiple HTTP categories are defined the first one which is (or relates to) to a Kind **MUST** be used for defining the type of the resource instance. Optional information which might be provided by the client and if available **MUST** be used are HTTP Links and HTTP X-OCCI-Attributes (mapping to Link and the attributes of an resource instance). Two ways can be used to create a new resource instance - HTTP POST or PUT:

```
> POST / HTTP/1.1
> [...]
>
> Category:compute;class=kind;scheme=http://schemas.ogf.org/occi/infrastructure#
> X-OCCI-Attribute:occi.compute.cores=2 occi.compute.hostname=foobar
> [...]

< HTTP/1.1 200 OK
< [...]
< Location:http://example.com/vms/user1/vm1
```

The path on which this POST verb is executed can be any existing path in the hierarchy of the Service provider's name-space. The OCCI implementation **MUST** return the Location of the newly created resource instance.

HTTP PUT can also be used to create a resource instance. In this case the client ask the service provider to create a resource instance at a certain path in the name-space hierarchy.³

```
> PUT /vms/user1/my_first_virtual_machine HTTP/1.1
> [...]
>
> Category:compute;class=kind;scheme=http://schemas.ogf.org/occi/infrastructure#
> X-OCCI-Attribute:occi.compute.cores=2 occi.compute.hostname=foobar
> [...]

< HTTP/1.1 200 OK
< [...]
```

The OCCI implementation will return an OK code.

Retrieving a resource instance For retrieval the HTTP GET verb is used. It **MUST** return at least the HTTP category which defines the Kind of the resource instance. HTTP Links pointing to related resource instances, other URI or Actions **MUST** be included if present. Only Actions currently applicable **SHOULD** be rendered using HTTP Links. The Attributes of the resource instance **MUST** be exposed to the client if available.

²The path **MUST** not end with an '/' - that would mean that a client operates on a path in the name-space hierarchy

³If a Service Provider does not want the user to define the path of a resource instance it can return a Bad Request return code - See section 3.4.5. Service Providers **MUST** ensure that the paths of REST resources stays unique in their name-space.

```

> GET /vms/user1/vm1 HTTP/1.1
> [...]

< HTTP/1.1 200 OK
< [...]
< Category:compute;class=kind;scheme=http://schemas.ogf.org/occi/infrastructure#
< Category:my_stuff;class=mixin;scheme=http://example.com/occi/my_stuff#
< X-OCCE-Attribute:occi.compute.cores=2 occe.compute.hostname=foobar
< Link: [...]

```

Updating a resource instance Before updating a resource instance it is RECOMMENDED that the client first retrieves the resource instance. Updating is done using the HTTP PUT verb. Only the information (HTTP Links, HTTP X-OCCE-Attributes or HTTP categories), which are updated MUST be provided along with the request.⁴

```

> PUT /vms/user1/vm1 HTTP/1.1
> [...]
>
> X-OCCE-Attribute: occe.compute.memory=4.0
> [...]

< HTTP/1.1 200 OK
< [...]

```

Deleting a resource instance A resource instance can be deleted using the HTTP DELETE verb. No other information SHOULD be added to the request.⁵

```

> DELETE /vms/user1/vm1 HTTP/1.1
> [...]

< HTTP/1.1 200 OK
< [...]

```

Triggering an Action on a resource instance To trigger an action on a resource instance the request MUST contain the HTTP Category defining the Action. It MAY include HTTP X-OCCE-Attributes which are the parameters of the action. Actions are triggered using the HTTP POST verb and by adding a query to the URI. This query exposes the term of the Action. If an action is not available a Bad Request should be returned.

```

> POST /vms/user1/vm1?action=stop HTTP/1.1
> [...]
> Category: Category:compute;class=action;scheme=...
> X-OCCE-Attribute:method=poweroff

< HTTP/1.1 200 OK
< [...]

```

3.2.2.2 Handling the Query Interface The query interface MUST be implemented by all OCCE implementations. It MUST be found at the path `/-/` off the root of the OCCE implementation. The following operations, listed below, MUST be implemented by the service.

Retrieval of all registered Kinds and Mixins The HTTP verb GET must be used to retrieve all IKinds and Mixins the service can handle. This allows the client to discover the capabilities of the OCCE implementation. The result MUST contain all information about the Kinds and Mixins (including Attributes and Actions assigned).

⁴Changing the type of the resource instance MUST not be possible.

⁵If the resource instances is a Link type the source and target must be updated accordingly

```

> GET /-/ HTTP/1.1
> [...]

< HTTP/1.1 200 OK
< [...]
< Category: compute;
    class=kind;
    scheme=http://schemas.ogf.org/occi/infrastructure#;
    attributes=occi.compute.cores,occi...;
    rel=http://schemas.ogf.org/occi/core\#entity;
    actions=http://schemas.ogf.org/occi/infrastructure/compute/action#stop,...;
    location=/compute;
< Category: my_stuff;
    class=mixin;
    scheme=http://example.com/occi/my_stuff#;
    location=/my_stuff;
< Category: storage;
    class=kind;
    scheme=http://schemas.ogf.org/occi/infrastructure#;
    attributes="...";
    actions="...";
    rel=http://schemas.ogf.org/occi/core\#entity;
    location=/storage;

```

An OCCI implementation **MUST** support a filtering mechanism. If a HTTP Category is provided in the request the server **MUST** only return the complete rendering of the requested Kind or Mixin.

Adding a Mixin definition To add a Mixin to the service the HTTP PUT verb **MUST** be used. All possible information for the Mixin must be defined. At least the HTTP Category term, scheme and location **MUST** be defined. Actions and Attributes are not supported:

```

> GET /-/ HTTP/1.1
> [...]
> Category: my_stuff;
    class=mixin
    scheme=http://example.com/occi/my_stuff#;
    rel=http://example.com/occi/something_else#mixin;
    location=/my_stuff;

< HTTP/1.1 200 OK
< [...]

```

The service might reject this request if it does not allow user-defined Mixins to be created. Also on name collisions of the defined location path the service provider might reject this operation.

Removing a Mixin definition A user defined Mixin **CAN** be removed (if allowed) by using the HTTP DELETE verb. The information about which Mixin should be deleted **MUST** be provided in the request:

```

> DELETE /-/ HTTP/1.1
> [...]
> Category: my_stuff;class=mixin;scheme=http://example.com/occi/my_stuff#;

< HTTP/1.1 200 OK
< [...]

```

3.2.2.3 Operations on Mixins or Kinds All the following operations CAN only be done on a location paths provided by Kinds and Mixins. It MUST end with an `/`.

Retrieving All Resource Instances Belonging to Mixin or Kind The HTTP verb GET must be used to retrieve all resource instances. The service provider MUST return a listing containing all resource instances which belong to the requested Mixin or Kind:

```
> GET /compute/ HTTP/1.1
> [...]

< HTTP/1.1 200 OK
< [...]
<
< X-OCCE-Location: http://example.com/vms/user1/vm1
< X-OCCE-Location: http://example.com/vms/user1/vm2
< X-OCCE-Location: http://example.com/vms/user2/vm1
```

Note: A OCCE implementation MUST support a filtering mechanism. If a HTTP Category is provided in the request the server MUST only return the resource instances belonging to the provided Kind or Mixin. The provided HTTP category definition SHOULD be different from the Kind or Mixin definition which defined the location path used in the request.

Triggering Actions on All Instances of a Mixin or Kind Actions can be triggered on all resource instances of the same Mixin or Kind. The HTTP POST verb MUST be used. Also the Action MUST be defined by the Kind or Mixin which defines the location path which is used in the request:

```
> POST /compute/;action=stop HTTP/1.1
> [...]
X-OCCE-Attribute:method=poweroff

< HTTP/1.1 200 OK
< [...]
```

Associate resource instances with Mixins One or multiple resource instances can be associated with a Mixin using the HTTP PUT verb. The URIs which uniquely defined the resource instance MUST be provided in the request:

```
> PUT /my_stuff/ HTTP/1.1
> [...]
> X-OCCE-Location:http://example.com/vms/user1/vm1,
    http://example.com/vms/user1/vm2,
    http://example.com/disks/user1/disk1

< HTTP/1.1 200 OK
< [...]
```

Unassociated resource instance(s) from a Mixin One or multiple resource instances can be removed from a Mixin using the HTTP DELETE verb. The URIs which uniquely defined the resource instance MUST be provided in the request:

```
> DELETE /my_stuff/ HTTP/1.1
> [...]
> X-OCCE-Location:http://example.com/vms/user1/vm1,
    http://example.com/vms/user1/vm2,
    http://example.com/disks/user1/disk1

< HTTP/1.1 200 OK
< [...]
```

3.2.2.4 Operation on Paths in the Name-space The following operations are defined when operating on paths in the name-space hierarchy which are not location paths nor resource instances. They MUST end with / (For example *http://example.com/vms/user1*).

Retrieving All resource instances Below a Path The HTTP verb GET must be used to retrieve all resource instances. The service provider MUST return a Listing containing all resource instances which are children of the provided URI in the name-space hierarchy:

```
> GET /vms/user1/ HTTP/1.1
> [...]

< HTTP/1.1 200 OK
< [...]
<
< X-OCCT-Location: http://example.com/vms/user1/vm1
< X-OCCT-Location: http://example.com/vms/user1/vm2
```

An OCCT implementations MUST support a filtering mechanism. If a category is provided in the request the server MUST only return the resource instances belonging to the provided Mixin or Kind.

Deletion of all resource instances below a path (Note: this is a potentially dangerous operation!)

The HTTP verb DELETE must be used to delete all resource instances under a hierarchy:

```
> DELETE /vms/user1/ HTTP/1.1
> [...]

< HTTP/1.1 200 OK
< [...]
```

3.3 Syntax and Semantics of the Rendering

3.3.1 Rendering of an OCCT-Category

The semantics of the HTTP category in the OCCT context is described in the OCCT Core & Models document. This rendering follows the Category header defined by the Web Categories specification ⁶ and MUST be rendered accordingly.

```
Category          = "Category" ":" category-value
category-value    = term ";" "class" "=" class-value *( ";" category-param )
class-value       = "action" | "mixin" | "kind"
category-param    = ( ( "scheme" "=" <> scheme "#" <> )
                    | ( category-extension ) )
category-extension = token [ "=" ( token | quoted-string ) ]
term              = token
scheme            = URI
```

3.3.2 Rendering of OCCT-Links and OCCT-Actions

The semantics of the HTTP Link header in the OCCT context is described in the OCCT Core & Models document. This rendering follows the Link header defined by the Web Linking specification ⁷ and MUST be rendered accordingly.

⁶<http://tools.ietf.org/html/draft-johnston-http-category-header-01>

⁷<http://tools.ietf.org/html/draft-nottingham-http-link-header-10>

```

Link           = "Link" ":" #link-value
  link-value   = "<" URI-Reference ">" *( ";" link-param )
  link-param   = ( ( "rel" "=" relation-types )
                  | ( link-extension ) )
  link-extension = token [ "=" ( token | quoted-string ) ]
  relation-types = relation-type
                  | <"> relation-type *( 1*SP relation-type ) <">
  relation-type = reg-rel-type | ext-rel-type
  reg-rel-type  = LOALPHA *( LOALPHA | DIGIT | "." | "-" )
  ext-rel-type  = URI

```

3.3.3 Rendering of OCCI-Attributes

The X-OCCI-Attribute MUST be used to render the attributes associated with a OCCI Kind. A simple key-value format is used. The field value consist of an attribute name followed by an equal sign ("=") and the attribute value. The attribute value must be double quoted if it includes a separator character, see RFC 2616 (page 16).

```

Attribute      = "X-OCCI-Attribute" ":" attribute-value
  attribute-value = token [ "=" ( token | quoted-string ) ]

```

Valid attribute names for OCCI Kinds are specified in appropriate Extension documents ??.

3.3.4 Rendering of Location-URLs

To render an OCCI representation solely in the header, the X-OCCI-Location HTTP header MUST be used to return a list of Kind URLs. Each header field value correspond to a single URL. Multiple Kind URLs are returned using multiple X-OCCI-Location headers. See RFC 2616 for information on how to render multiple HTTP headers.

```

Location       = "X-OCCI-Location" ":" location-value
  location-value = "<" URI ">" *(", " URI)

```

3.4 General HTTP Behaviors Adopted by OCCI

The following sections deal with some general HTTP features which are adopted by OCCI.

3.4.1 Security and Authentication

OCCI does not require that an authentication mechanism be used nor does it require that client to service communications are secured. It does recommend that an authentication mechanism be used and that where appropriate, communications are encrypted using HTTP over TLS. The authentication mechanisms that CAN be used with OCCI are those that can be used with HTTP and TLS.

3.4.2 Caching Headers

The responses from an OCCI implementation CAN include headers with strict Caching purposes like E-Tags.

3.4.3 Versioning

Information about what version of OCCI is supported by a OCCI implementation MUST be advertised to a client on each response to a client. 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 specification. In the case of a HTTP Header Rendering, the server response MUST relay versioning information using the HTTP header name 'Server'.

```
HTTP/1.1 202 Accepted
Server: occi-server/1.1 (linux) OCCI/1.1
[...]
```

Complimenting the service-side behavior of an OCCI implementation, a client SHOULD indicate to the OCCI service implementation the version it expects to interact with. For the clients, the information SHOULD be advertised in all requests it issues. A client request SHOULD relay versioning information in the 'User-Agent' header. The 'User-Agent' field MUST include the same value (OCCI/X.Y) as supported by the Server HTTP header.

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

If a OCCI implementation receives a request from a client that supplies a version number higher than the service supports, the service MUST respond back to the client with an exception indicating that the requested version is not implemented. Where a client implements OCCI using a HTTP transport, the HTTP code 501, not implemented, MUST be used.

OCCI implementations which implement this version of the Document MUST use the version String *OCCI/1.1*.

3.4.4 Content-type and Accept headers

A server MUST react according to the Accept header the client provides. If none is given - or */* is used - the service MUST use the Content-type *text/plain*. This is the fall-back rendering and MUST be implemented. Otherwise the according rendering MUST be used. Each Rendering SHOULD expose which Accept and Content-type header fields it can handle. Overall the service MUST support the *text/occi* and *text/plain* Content-types.

The server MUST also return the proper Content-type header. If a client provides information with a Content-Type - the information MUST be parsed accordingly.

When the Client request a Content-Type that will result in an incomplete or faulty rendering the Service MUST return the unsupported media type , 415, HTTP code.

The following examples demonstrate the behavior of an HTTP GET operations on the resource instance using two different HTTP Accept headers:

```
> GET /vms/user1/vm1 HTTP/1.1
> Accept: text/plain
> [...]

< HTTP/1.1 200 OK
< [...]
< Category:compute;class=kind;scheme=http://schemas.ogf.org/occi/infrastructure#
< Category:my_stuff;class=mixin;scheme=http://example.com/occi/my_stuff#
< X-OCCI-Attribute:occi.compute.cores=2 occi.compute.hostname=foobar
< Link: [...]
```

And with *text/occi* as HTTP Accept header:

```
> GET /vms/user1/vm1 HTTP/1.1
> Accept: text/occi
> [...]

< HTTP/1.1 200 OK
< Category:compute;scheme=http://schemas.ogf.org/occi/infrastructure#
< Category:my_stuff;scheme=http://example.com/occi/my_stuff#
< X-OCCEI-Attribute:occi.compute.cores=2 occi.compute.hostname=foobar
< Link: [...]
< [...]
OK
```

3.4.4.1 The Content-type *text/plain* While using this rendering with the Content-Type *text/plain* the information described in section 3.3 MUST be placed in the HTTP Body.

Each rendering of an OCCI base type will be placed in the body. Each entry consists of a name followed by a colon (":") and the field value. The format of the field value is specified separately for each of the three header fields, see section 3.3.

3.4.4.2 The Content-type *text/occi* While using this rendering with the Content-Type *text/occi* the information described in section 3.3 MUST be placed in the HTTP Header. The body MUST contain the string 'OK' on successful operations.

The HTTP header fields MUST follow the specification in RFC 2616 [7]. A header field consists of a name followed by a colon (":") and the field value. The format of the field value is specified separately for each of the header fields, see section 3.3.

Limitations: HTTP header fields MAY appear multiple times in a HTTP request or response. In order to be OCCI compliant the specification of multiple message-header fields according to RFC 2616 MUST be fully supported. In essence there are two valid representation of multiple HTTP header field values. A header field might either appear several times or as a single header field with a comma-separated list of field values. Due to implementation issues in many web frameworks and client libraries it is RECOMMENDED to use the comma-separated list format for best interoperability.

HTTP header field values which contain separator characters MUST be properly quoted according to RFC 2616.

Space in the HTTP header section of a HTTP request is a limited resource. By this, it is noted that many HTTP servers limit the number of bytes that can be placed in the HTTP Header area. Implementers MUST be aware of this limitation in their own implementation and take appropriate measures so that truncation of header data does NOT occur.

3.4.4.3 The Content-type *text/uri-list* This Rendering can handle the *text/uri-list* Accept Header. It will use the Content-type *text/uri-list*.

This rendering cannot render resource instances or Kinds or Mixins directly but just links to them. For concrete rendering of Kinds and Categories the Content-types *text/occi*, *text/plain* MUST be used. If a request is done with the *text/uri-list* in the Accept header, while not requesting for a Listing a Bad Request MUST be returned.

3.4.5 Return codes

At any point the service provider CAN return any of the following HTTP Return Codes:

Table 2. HTTP Return Codes

Code	Description	Notes
200	OK	
202	Accepted	
400	Bad Request	Used for asynchronous non-blocking calls.
401	Unauthorized	For example on parsing errors or missing information
403	Forbidden	
405	Method Not Allowed	
409	Conflict	
410	Gone	
415	Unsupported Media Type	
500	Internal Server Error	
501	Not Implemented	
503	Service Unavailable	

3.5 More complete examples

Since most examples are not complete due to space limitations this section will give some more complete examples.

3.5.1 Creating a resource instance

```
> POST / HTTP/1.1
> User-Agent: curl/7.21.0 (x86_64-pc-linux-gnu) libcurl/7.21.0 OpenSSL/0.9.8o zlib/1.2.3.4 libidn/1.
> Host: localhost:8080
> Accept: */*
> Cookie: pyocci_user=Zm9v|1291753962|7011a4821179ff98ea96d4b44fade0512b1ffc52
> Content-Type: text/occi
> Category: compute;class=kind;scheme=http://schemas.ogf.org/occi/infrastructure#
>
< HTTP/1.1 200 OK
< Content-Length: 2
< Content-Type: text/html; charset=UTF-8
< Location: /users/foo/compute/b9ff813e-fee5-4a9d-b839-673f39746096
< Server: pyocci OCCI/1.1
<
* Connection #0 to host localhost left intact
* Closing connection #0
OK%
```

3.5.2 Retrieving a resource instance

```
> GET /users/foo/compute/b9ff813e-fee5-4a9d-b839-673f39746096 HTTP/1.1
> User-Agent: curl/7.21.0 (x86_64-pc-linux-gnu) libcurl/7.21.0 OpenSSL/0.9.8o zlib/1.2.3.4 libidn/1.
> Host: localhost:8080
> Accept: */*
> Cookie: pyocci_user=Zm9v|1291753962|7011a4821179ff98ea96d4b44fade0512b1ffc52
>
< HTTP/1.1 200 OK
< Content-Length: 415
< Etag: "ef485dc7066745cb0fe1e31ecdd4895c356b5bd5"
< Content-Type: text/plain
< Server: pyocci OCCI/1.1
<
Category: compute;class=kind;scheme=http://schemas.ogf.org/occi/infrastructure#
```

Link: </users/foo/compute/b9ff813e-fee5-4a9d-b839-673f39746096?action=start>
 X-OCCE-Attribute: occi.compute.architecture=x86
 X-OCCE-Attribute: occi.compute.state=inactive
 X-OCCE-Attribute: occi.compute.speed=1
 X-OCCE-Attribute: occi.compute.memory=2
 X-OCCE-Attribute: occi.compute.cores=2
 X-OCCE-Attribute: occi.compute.hostname=dummy

4 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
Andy Edmonds	Intel - SLA@SOI project	andy at edmonds.be
Sam Johnston	Google	samj at samj.net
Gary Mazzaferro	OCCE Counselour - Exxia, Inc.	garymazzaferro at gmail.com
Thijs Metsch	Platform Computing, Sun Microsystems	tmetsch at platform.com
Ralf Nyrén	Aurenav	ralf at nyren.net
Alexander Papaspyrou	TU-Dortmund	alexander.papaspyrou at tu-dortmund.de
Shlomo Swidler	Orchestratus	shlomo.swidler at orchestratus.com

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

5 Glossary

Term	Description
Action	An OCCI base type. Represent an invocable operation on a Entity sub-type instance or collection thereof.
Category	A type in the OCCI model. The parent type of Kind.
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.
Kind	A type in the OCCI model. A core component of the OCCI classification system.
Link	An OCCI base type. A Link instance associate one Resource instance with another.
mixin	An instance of the Mixin type associated with a resource instance . The “mixin” concept as used by OCCI <i>only</i> applies to instances, never to Entity types.
Mixin	A type in the OCCI model. A core component of the OCCI classification system.
OCCI	Open Cloud Computing Interface
OCCI base type	One of Entity, Resource, Link or Action.
OGF	Open Grid Forum
Resource	An OCCI base type. The parent type for all domain-specific resource types.
resource instance	An instance of a sub-type of Entity. The OCCI model defines two sub-types of Entity, the Resource type and the Link type. However, the term <i>resource instance</i> is defined to include any instance of a <i>sub-type</i> of Resource or Link as well.
Tag	A Mixin instance with no attributes or actions defined.
Template	A Mixin instance which if associated at resource instantiation time pre-populate certain attributes.
type	One of the types defined by the OCCI model. The OCCI model types are Category, 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

6 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.

7 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.

8 Full Copyright Notice

Copyright ©Open Grid Forum (2009-2010). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the OGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.

9 References

References

- [1] T. Metsch, A. Edmonds, and R. Nyrén, “Open Cloud Computing Interface – Core,” <http://ogf.org/gf/docs/>, Nov. 2010, in Public Comment.
- [2] T. Metsch and A. Edmonds, “Open Cloud Computing Interface – HTTP Rendering,” <http://ogf.org/gf/docs/>, Nov. 2010, in Public Comment.
- [3] —, “Open Cloud Computing Interface – Infrastructure,” <http://ogf.org/gf/docs/>, Nov. 2010, in Public Comment.
- [4] 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>
- [5] 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>
- [6] T. R. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, University of California, Irvine, 2000.
- [7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1,” RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999, updated by RFCs 2817, 5785. [Online]. Available: <http://www.ietf.org/rfc/rfc2616.txt>