Open Cloud Computing Interface - Core & Models

Status of This Document

This document provides information to the Cloud and Grid community. Distribution is unlimited.

Abstract

This document is part of the Open Cloud Computing Interface (OCCI) specification document series. The OCCI document series describes what each OCCI compatible interface needs to provide. The overall OCCI specification itself is setup modular to be extensbible and includes the following parts:

- The OCCI Core & Models
- The OCCI Infrastructure Models
- OCCI XHTML5 rendering
- OCCI HTTP Header rendering

Each of these parts is described in a separate document so the overall specification comes in the form of a document series. Where as this document describes the OCCI Core & models.

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

1. OCCI Core Specification & Models

1.1. Introduction

The Open Cloud Computing Interface (OCCI) is an open protocol for all cloud computing services. As a RESTful interface, it deviates from the underlying HyperText Transfer Protocol (HTTP) only where absolutely necessary and can be described as a "Resource Oriented Architecture (ROA)".*RWS* Unlike other envelope-based protocols which operate in-band, all existing HTTP features are available for caching, proxying, gatewaying and other advanced functionality such as partial GETs.

Each resource is identified by URL(s) and has one or more native representations as well as an XHTML5 rendering for direct end-user accessibility with embedded semantic web markup. As such OCCI can present both a machine interface (using native resource renderings) and a user interface (using HTML markup with forms and other web technologies such as Javascript/Ajax). HTTP content negotiation is used to select between alternative

representations and metadata including associations between resources is exposed via HTTP headers (e.g. the `Link:` and `Category:` headers).

In this way OCCI is not responsible for the representations themselves, rather it enables users to organise and group resources together to build arbitrarily complex systems of inter-related resources. It relies on existing standards for rendering and does not make any recommendations of one standard format over any other.

Tip

This is the case for the World Wide Web today where many image, video and other supporting formats co-exist. Browsers support a number of the common formats and users choose the most appropriate for the task.

1.1.1. Example

```
> GET /us-east/webapp/vm01 HTTP/1.1
> User-Agent: occi-client/1.0 (linux) libcurl/7.19.4 OCCI/1.0
> Host: cloud.example.com
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: application/ovf
< Link: </us-east/webapp/vm01;start>;
<       rel="http://purl.org/occi/action#start";
<       title="Start"
< Link: </us-east/webapp/build.pdf>;
<       rel="related";
<       title="Documentation";
<       type="application/pdf"
< Category: compute;
<       label="Compute Resource";
<       scheme="http://purl.org/occi/kind#"
< Server: occi-server/1.0 (linux) OCCI/1.0
< Connection: close
<
< <?xml version="1.0" encoding="UTF-8"?>
...
```

1.2. Essentials

1.2.1. Connection

1.2.1.1. Authentication

Servers *may* require that requests be authenticated using standard HTTP-based authentication mechanisms (including OAuth).*OAuth* They indicate this requirement by returning `HTTP 401` with a `WWW-Authenticate` header and a suitable challenge (e.g. `Basic`, `Digest`, `OAuth`). The client then includes appropriate `Authorization` headers in its responses.*RFC2617*

Servers *may* set and clients *may* accept *cookies* in order to maintain authentication state between requests. Such sessions *should not* be used for other purposes (such as server-side state) in line with RESTful principles.*RFC2109*

1.2.1.2. Versioning

Servers and clients *should* indicate the latest version of OCCI they support (e.g. `1.0`) by way of the `Server:` and `User-Agent:` headers respectively, using the token "`OCCI`" (e.g.

"`OCCI/1.0`"). If none is provided the latest (highest supported version number) available version offered by the server *should* be used.

1.2.2. Addressing

The entry point is defined by a single URL which may be a collection of resources or some other page as defined by the implementor (e.g. a landing page). All resources *must* be addressible by URLs (whose structure is opaque and at the discretion of the implementor) and discoverable via search and/or link traversal from the entry point.

Tip

Clients will typically conduct a `GET` or `HEAD` request on the root ("`/`") and discover the category search interface, from which they can learn the supported categories/ kinds and retrieve some or all of them.

Example 1. Example entry point

Retrieve a collection of desired resources (having already discovered the category search URL and available categories):

```
> GET /-/compute HTTP/1.1
> Accept: text/uri-list
>
< HTTP/1.1 200 OK
< Content-type: text/uri-list
<
< /node1
< /node2
```

Tip

This discovery mechanism was selected so as to be compatible with existing content hosted at the same URL (e.g. `http://cloud.example.com`).

1.3. Operations

1.3.1. HTTP Verbs

Create, Retrieve, Update and Delete (CRUD) operations map to the POST, GET, PUT and DELETE HTTP verbs respectively. HEAD and OPTIONS verbs may be used to retrieve metadata and valid operations without the entity body to improve performance. WebDAV definitions are used for MKCOL, MOVE and COPY.

Warning

Some providers may implement a subset of these operations, and those available to you for a given resource (if any) may depend on security policy. Be prepared to handle exceptions if you attempt to call operations that are not available to you.

POST (Create)	"The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line."*RFC2616* POSTing a representation (e.g. OVF) to a collection (e.g. / compute) will result in a new resource being created (e.g. /

compute/123) and returned in the Location: header. POST is also used with HTML form data to trigger verbs (e.g. restart)

GET (Retrieve - Metadata and Entity)	"The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI."*RFC2616* GETting a resource (e.g. /compute/123) will return a representation of that resource in the most appropriate supported format specified by the client in the Accept header. Otherwise "406 Not Acceptable" will be returned.
PUT (Create or Update)	"The PUT method requests that the enclosed entity be stored under the supplied Request-URI."*RFC2616* PUTting a representation (e.g. OVF) to a URL (e.g. /compute/123) will result in the resource being created or updated. The URL is known or selected by the client, in contrast to POSTs where the URL is selected by the server.
DELETE (Delete)	"The DELETE method requests that the origin server delete the resource identified by the Request-URI."*RFC2616* DELETE results in the deletion of the resource (and everything "under" it, as appropriate).

Tip

It is possible to instruct the server to create a resource based on a default configuration (without requiring client support) by doing an empty POST/PUT, specifying "`Content-type: application/occi`" (such that the web server knows where to route the request) and specifying the appropriate *`kind`* category (such that OCCI knows what to create).

Additionally the following HTTP methods are used:

COPY (Duplicate)	"The COPY method creates a duplicate of the source resource identified by the Request-URI, in the destination resource identified by the URI in the Destination header."*RFC4918*
HEAD (Retrieve - Metadata Only)	"The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response."*RFC2616*
MKCOL (Make Collection)	"MKCOL creates a new collection resource at the location specified by the Request-URI."*RFC4918*
MOVE (Relocate)	"The MOVE operation on a non-collection resource is the logical equivalent of a copy (COPY), followed by consistency maintenance processing, followed by a delete of the source, where all three actions are performed in a single operation."*RFC4918*
OPTIONS	"The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI."*RFC2616*

Tip

Implementors may offer full WebDAV support in order to allow clients to enumerate the entire tree, interact with the resources via standard file managers (e.g. Windows Explorer, Mac OS X Finder), etc.

1.3.2. Actions

An *action* is some process that can be carried out on one or more *resource*s, which may result in a state change and/or the creation of new resource(s).

Tip

Use common sense to decide what functionality should be exposed by way of actions and consult the list of existing actions and verbs before creating your own. For example it does not make sense to resize a storage resource by setting the “size” attribute (indeed there may not be space available or the filesystem may not support resizing and in any case the operation will take some time), nor to start a machine by changing the state from “stopped” to “running”.

Each available *action* for a given *resource* is indicated via a *link* with `class` extension set to “`action`” (such that clients can identify actions, including those from third-parties, without deriving meaning from the `rel` URI).

```
Link: </us-east/webapp/vm01;start>;
      rel="http://purl.org/occi/action#start";
      class="action";
      title="Start"
```

Actions defined by this standard reside under the `http://purl.org/occi/action#` namespace but anyone can define a new action by allocating a URI they control.

Warning

Defining your own actions can lead to interoperability problems and should be a last resort reserved for unique functionality. A simple peer review process is available for extending the registries which should be used where possible.

An *action* is triggered via an HTTP POST and depending on the action requested (e.g. `resize`), parameters *may* be provided using HTML forms (e.g. `application/x-www-form-urlencoded`). In the case of HTML-based renderings the actions can therefore be actual HTML forms.

Tip

Some resources can be interacted with but not rendered due to the nature of the resource or prevailing security policies (for example, an operator may be able to backup a machine without knowing anything about it).

1.3.2.1. Asynchronous Actions

Synchronous actions *may* return `200 OK` on successful completion or `201 Created` with a `Location:` header indicating a new resource for audit purposes.

Tip

Assume that clients are paranoid and want audit trails for all but the most trivial of actions.

In the event that the *action* does not complete immediately it *should* return `HTTP 202 Accepted` and a `Location:` header indicating a new resource where status and other pertinent information can be obtained.

Tip

Don't keep clients waiting - if you're not sure to return immediately then give them a resource they can monitor. For example by responding with an 202 Accepted return code and include a location: header, as described.

1.3.2.2. Advanced Actions

The specific parameters required and allowable values for them depend on the action and for advanced actions *may* require sending of custom *content type*s rather than `application/x-www-form-encoded`.

1.3.2.3. State Machines

State machines are maintained on the server side and possible transitions are advertised to the client by way of action links. The links offered to a given client may depend on the resource, its current state, security policy, etc.

Tip

Many state transitions will not be effected immediately so be prepared to handle asynchronous responses.

1.4. Model

The model defines the objects and how they interrelate. An interface exposes "kinds" which have "attributes" and on which "actions" can be performed. The attributes are exposed as key-value pairs and applicable actions as links, following the REST hypertext constraint (whereby state transitions are defined *in-band* rather than via rules).

1.4.1. Kinds

Each category of resources distinguished by some common characteristic or quality is called a *kind* (e.g. `compute`, `network`, `storage`, `queue`, `application`, `contact`).

Kinds defined by this standard live in the `http://purl.org/occi/kind/` namespace but anyone can define a new kind by using a URI they control as the term.

Warning

Defining your own kinds can lead to interoperability problems and should be a last resort reserved for unique functionality. A simple peer review process is available for extending the registries which should be used where possible.

Each resource *must* specify a kind by way of a *category* within the *scheme* "`http://purl.org/occi/kind#`".

Tip

The word *type* is not used in this context in order to avoid confusion with Internet media types.

1.4.1.1. Attributes

An *attribute* is a specification that defines a property of an object. It is expressed in the form of key-value pairs. Attributes are divided into namespaces which are separated by the dot character (".").

Tip

This scalable approach was derived from the Mozilla Firefox `about:config` page.

Attributes defined by this standard reside at the root but anyone can define a new attribute by allocating a unique namespace based on their reversed Internet domain (e.g. "`com.example.attribute`").

Warning

Defining your own attributes can lead to interoperability problems and should be a last resort reserved for unique functionality. A simple peer review process is available for extending the registries which should be used where possible.

Registry Entries

Table 1. Core Attributes

Attribute	Description	Type	Example
`id`	Immutable, unique identifier for the resource	URI	`urn:uuid:d0e9f0d0-f62d-4f28-bc90-23b0bd871770` or `urn:aws:ami-123456`
`title`	Display name for the resource	String	`Compute Resource #123`
`summary`	Description of the resource	String	`A virtual compute resource`

1.4.2. Categories

Category information allows for flexible organisation of resources into one or more vocabularies (each of which is referred to as a *scheme*).

The category model was derived from Atom and consists of three attributes:

term — The term itself (e.g. "`compute`")

scheme (optional) — The vocabulary (e.g. "`http://purl.org/occi/kind#`")

label (optional)	A human-friendly display name for the term (e.g. “`Compute Resource`”)

Category schemes and/or terms defined by this standard reside throughout the `http://purl.org/occi/` namespace but anyone can define a new scheme by allocating a URI they control.

Tip

Categories provide a flexible way to manage resources by taxonomy (categories) and/or folksonomy (tags), where both can be shared between [groups of] users or globally. For example, users can create schemes for resource locations (e.g. `US-East`, `US-West`, `Europe`), operating systems (e.g. `Windows`, `Linux`) and patch levels (e.g.

Example 2. Category examples

OCCI kinds are represented by a category:

```
Category: compute;
      label="Compute Resource";
      scheme="http://purl.org/occi/kind#"
```

Implementors and users can also define their own vocabularies by defining schemes:

```
Category: cluster1;
      label="Cluster #1";
      scheme="http://example.com/clusters#"
```

1.4.2.1. Querying

The category query interface can be accessed by constructing an URL with the desired categories added to the path. Categories can be negated by prefixing with “-” and schemes may be specified with braces.

Example 3. Example category query

Locate the category search root (which *should* be `/-/`):

```
> HEAD / HTTP/1.1
>
< HTTP/1.1 200 OK
< Link: </-/>; rel="search"; title="Category Search"
```

Discover the available categories (which will all be returned in the same format as they appear in the HTTP headers):

```
> GET /-/ HTTP/1.1
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-type: application/occi
<
< Category: compute; label="Compute Resource"; scheme="http://
purl.org/occi/kind#"
< Category: network; label="Network Resource"; scheme="http://
purl.org/occi/kind#"
< Category: storage; label="Storage Resource"; scheme="http://
purl.org/occi/kind#"
< Category: us-east; label="US East Coast"; scheme="http://
example.com/locations"
< Category: us-west; label="US West Coast"; scheme="http://
example.com/locations"
< Category: demo; label="My Customer Demo"; scheme="http://
example.com/~user/"
```

Query the category search interface for the desired category(s):

```
> GET /-/compute/us-west HTTP/1.1
> Accept: text/uri-list
>
< HTTP/1.1 200 OK
< Content-type: text/uri-list
<
< /vm01
< /webapp/web01
< /webapp/web02
< /webapp/db01
```

1.4.2.2. Registry Entries

Table 2. Core Category Schemes

Scheme	Description	Example
`http://purl.org/occi/kind#`	OCCI Kinds	`compute`
`http://purl.org/occi/category#countries`	ISO Country Codes	`us`
`http://purl.org/occi/category#us-states`	US States	`ca`
`http://purl.org/occi/category#operating-systems`	Operating Systems	`linux`

Scheme	Description	Example
`http://purl.org/occi/category#regulations`	Regulation compliance	`sox`

Other categories schemes can be added to the registry.

1.4.3. Collections

Where an operation could return multiple resources (e.g. categories, searches) this is referred to as a *collection*. Collections are returned as a list of URLs in `text/uri-list` format.*RFC2483*

Tip

Collections are passed by reference for simplicity rather than performance reasons, requiring O(n+1) requests. Including metadata (via a wrapper format like Atom or SOAP) and/or the data itself would provide O(1) performance, though this "pass by value" approach should only be considered where the representations are known to be small as encoding adds significant overhead.

Example 4. Example collection

```
# OCCI Example Collection
/examples/custom-extension
/examples/lamp-multi-vm
/examples/lamp
/examples/myservice
```

1.4.3.1. Paging

Collections *may* be divided into *page*s, with each linking to the “first”, “last”, “next” and “previous” *link relation*s. The required `class` extension, with the value of `paging`, allows clients to group links together in the user interface and the server to specify e.g. "Next 10", "Next 100", etc.

```
Link: <http://example.com/xyz;start=0>; rel="first"; title="First";
 class="paging"
Link: <http://example.com/xyz;start=400>; rel="previous";
 title="Previous"; class="paging"
Link: <http://example.com/xyz;start=500>; rel="self"; title="Self";
 class="paging"
Link: <http://example.com/xyz;start=600>; rel="next"; title="Next";
 class="paging"
Link: <http://example.com/xyz;start=900>; rel="last"; title="Last";
 class="paging"
```

1.4.4. Linking

Web linking standards for HTTP [*LINK*] and HTML [*HTML5*] are used to indicate associations between resources. All formats *must* support *in-band* linking including:

- Link relations (e.g. `rel="alternate"`)
- Pointers to resources (e.g. `href="http://example.com/"`)
- Internet media types (e.g. `type="text/html"`)
- Extensibility (e.g. `attribute="value"`)

```
Link: </us-east/webapp/build.pdf>;
      rel="related";
      title="Documentation";
      type="application/pdf"
```

*Link relation*s defined by this standard reside under the `http://purl.org/occi/rel` namespace but anyone can define a new *link relation* by allocating a URI they control.

1.4.4.1. Registry Entries

Table 3. Core Link Relations

Relation	Description
`first`	“An IRI that refers to the furthest preceding resource in a series of resources.” [*LINK*]
`help`	“The referenced document provides further help information for the page as a whole.” [*HTML5*]
`icon`	“The specified resource is an icon representing the page or site, and should be used by the user agent when representing the page in the user interface.” [*HTML5*]
`last`	“An IRI that refers to the furthest following resource in a series of resources.” [*LINK*]
`next`	“A URI that refers to the immediately following document in a series of documents.” [*LINK*]
`previous`	“A URI that refers to the immediately preceding document in a series of documents.” [*LINK*]
`search`	“The referenced document provides an interface specifically for searching the document and its related resources.” [*HTML5*, *OpenSearch*]
`self`	“Identifies a resource equivalent to the containing element” [*RFC4287*]

1.5. Extensibility

The interface is fully extensible, both via a public peer review process (in order to update the specification itself, usually via registries) and via independent allocation of unique namespaces (in order to cater for vendor-specific enhancements).

1.5.1. Foreign markup

Implementations *must* accept and forward but otherwise ignore markup they do not understand.

1.6. Security Considerations

Encryption is not required by the specification in order to cater for sites that do not or can not use it (e.g. due to export restrictions, performance reasons, etc.), however SSL/TLS *should* be used over public networks including the Internet.

Glossary

in-band	“Sending of metadata and control information in the same band, on the same channel, as used for data”, for example, by embedding it in HTML. [`http://en.wikipedia.org/wiki/In-band`]
kind	“A category of things distinguished by some common characteristic or quality”, for example events, messages,

	media. [http://wordnetweb.princeton.edu/perl/webwn?s=kind]
out-of-band	"Communications which occur outside of a previously established communications method or channel", for example, in HTTP headers. [http://en.wikipedia.org/wiki/Out-of-band_signaling]
type	Internet media (MIME) type.

Bibliography

Normative References

[RFC2109] *RFC 2109 - HTTP State Management Mechanism*. http://tools.ietf.org/html/rfc2109. Internet Engineering Task Force (IETF) 1997-02.

[RFC2483] *RFC 2483 - URI Resolution Services Necessary for URN Resolution*. http://tools.ietf.org/html/rfc2483#section-5 [http://tools.ietf.org/html/rfc2109]. Internet Engineering Task Force (IETF) 1999-01.

[RFC2616] *RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1*. http://tools.ietf.org/html/rfc2616. Internet Engineering Task Force (IETF) 1999-06.

[RFC2617] *RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication*. http://tools.ietf.org/html/rfc2617 [http://tools.ietf.org/html/rfc2616]. Internet Engineering Task Force (IETF) 1999-06.

[RFC3339] *RFC 3339 - Date and Time on the Internet: Timestamps*. http://tools.ietf.org/html/rfc3339. Internet Engineering Task Force (IETF) 2002-07.

[RFC4918] *RFC 4918 - HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)*. http://tools.ietf.org/html/rfc4918 [http://tools.ietf.org/html/rfc2616]. Internet Engineering Task Force (IETF) 2007-06.

[OpenSearch] *OpenSearch 1.1*. http://www.opensearch.org/Specifications/OpenSearch/1.1 [http://tools.ietf.org/html/rfc2616]. A9.com, Inc. (an Amazon company) Clinton DeWitt. Joel Tesler . Michael Fagan . Joe Gregorio . Aaron Sauve . James Snell . 2009.

Informative References

[RFC4287] *RFC 4287 - The Atom Syndication Format*. http://tools.ietf.org/html/rfc4287. Robert Syre. Mark Nottingham. Internet Engineering Task Force (IETF) 2005-12.

[HTML5] *HTML 5*. http://www.w3.org/TR/html5/ [http://tools.ietf.org/html/rfc4287]. Ian Hickson. David Hyatt. World Wide Web Consortium (W3C) 2009-08-25.

[OAuth] *OAuth*. http://oauth.net/core/1.0 [http://tools.ietf.org/html/rfc2616]. OAuth Core Workgroup <spec@oauth.net>. 2007-12-04.

[RWS] *RESTful Web Services*. http://oreilly.com/catalog/9780596529260/. 9780596529260. O'Reilly Media Leonard Richardson. Sam Ruby. 2007-05.

[LINK] *Web Linking*. http://tools.ietf.org/html/draft-nottingham-http-link-header. Internet Engineering Task Force (IETF) Mark Nottingham. 2009-07-12.

[CATEGORY] *Web Categories*. http://tools.ietf.org/html/draft-johnston-http-category-header. Internet Engineering Task Force (IETF) Sam Johnston. 2009-07-1.

2. Contributors

The following people have contributed to this document.

Table 4. List of contributors

Name	Affiliation	Contact
Andy Edmonds	Intel - SLA@SOI project	andy@edmonds.be
Sam Johnston	Australian Online Solutions	samj@samj.net
Thijs Metsch	Sun Microsystems - RESERVOIR project	thijs.metsch@sun.com
Gary Mazzaferro	OCCI Counselour - Exxia, Inc.	garymazzaferro@gmail.com

3. 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.

4. 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.

5. Full Copyright Notice