
OCCI Core Specification

Introduction

The Open Cloud Computing Interface is an open protocol for cloud computing services. It is as close as possible to the underlying HyperText Transfer Protocol (HTTP), deviating only where absolutely necessary, and can be described as a "Resource Oriented Architecture (ROA)". RWS All existing HTTP features are available for caching, proxying, gatewaying and other advanced functionality.

Each resource (identified by a canonical URL) has one or more representations which may or may not be hypertext (e.g. HTML). Metadata including associations between resources is exposed via HTTP headers (e.g. the Link: header).

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 and relies on existing standards for rendering. HTTP content negotiation is used to select between alternative representations, which can also be advertised by way of links.

```
> GET /us-east/webapp/vm01 HTTP/1.1
> User-Agent: occi-client/1.0 (linux) libcurl/7.19.4 OpenSSL/0.9.8k zlib/1.2.3
> Host: cloud.example.com
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sat, 10 Oct 2009 12:56:51 GMT
< Expires: -1
< Cache-Control: private, max-age=0
< 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
< Connection: close
<
< <?xml version="1.0" encoding="UTF-8"?>
< <Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<       xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"
<       xmlns="http://schemas.dmtf.org/ovf/envelope/1"
<       xml:lang="en-US">
...

```

Basics

URL Namespace

The interface is defined by a single URL entry point which will either be a *collection*, contain *link(s)* to *collection(s)* or both.

Kinds, Actions and Attributes

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).

CRUD Operations

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.

POST (Create)	<p>“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</p> <p>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)</p>
GET (Retrieve - Metadata and Entity)	<p>“The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI.”RFC2616</p> <p>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.</p>
PUT (Create or Update)	<p>“The PUT method requests that the enclosed entity be stored under the supplied Request-URI.”RFC2616</p> <p>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 which case UUIDs should be used), in contrast to POSTs where the URL is selected by the server.</p>
DELETE (Delete)	<p>“The DELETE method requests that the origin server delete the resource identified by the Request-URI.”RFC2616</p> <p>DELETE results in the deletion of the resource (and everything "under" it, as appropriate).</p>

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 main-

tenance 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

Connection

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 in line with RESTful principles.RFC2109

Versioning

Every request *should* include an `occi.version` attribute indicating the version of the API requested (e.g. 1.0). If none is provided the latest available version *shall* be used.

Model

The model defines the objects themselves without regard to how they interrelate.

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 allocating a URI they control.

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.

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 under the `occi` namespace (e.g. "`occi.abc`") but anyone can define a new attribute by allocating a unique namespace based on their reversed Internet domain (e.g. "`com.cisco.cdp`"). A number of attributes are common to all *kinds*.

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.

Actions

An *action* is some process that can be carried out on one or more *resources*.

Each available *action* for a given *resource* is indicated via a *link* with the `action` class.

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

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.

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 types* rather than `application/x-www-form-encoded`.

Meta-model

The meta-model defines how objects interrelate.

Categories

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

The meta-model was derived from Atom, consisting of three attributes:

term	The term itself (e.g. "compute")
scheme (optional)	The vocabulary (e.g. " <code>http://purl.org/occi/kind/</code> ")
label (optional)	A human-friendly display name for the term (e.g. "Compute Resource")

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

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.

Querying

TODO: Pull query interface from GData: <http://code.google.com/apis/gdata/docs/2.0/reference.html#Queries>

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 links in `text/uri-list` format.

Tip

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

Any given URL can be a collection and/or advertise *links* to other *collections* using the `collection` class.

Tip

The root (“”) *should* expose collections *in-band* and/or *out-of-band* in order for clients to discover resources.

```
Link: <http://example.com/123/audit>;  
      rel="http://purl.org/occi/collection/audit";  
      title="Audit Entries"
```

Paging

Collections *may* be divided into *pages*, with each linking to the “first”, “last”, “next” and “previous” *link relations*.

```
Link: <http://example.com/xyz;start=0>; rel="first"  
Link: <http://example.com/xyz;start=400>; rel="previous"  
Link: <http://example.com/xyz;start=500>; rel="self"  
Link: <http://example.com/xyz;start=600>; rel="next"  
Link: <http://example.com/xyz;start=900>; rel="last"
```

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 relations 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.

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).

Foreign markup

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

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.
- [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.