OCCI Core Specification

Introduction

The Open Cloud Computing Interface is an open community consensus API, initially targeting cloud infrastructure services or "Infrastructure as a Service (IaaS)". A "Resource Oriented Architecture (ROA)", it is as close as possible to the underlying HyperText Transfer Protocol (HTTP), deviating only where absolutely necessary. Each resource (identified by a canonical URL) has zero 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), except in the case of collections where Atom is used as the meta-model.

Tip

Some resources can be interacted with but not rendered due to the nature of the resource or prevailing security policies.

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) (*in-band* and/or *out-of-band*) 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 HATEOAS principles (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 MOVE and COPY. All existing HTTP features is available for caching, proxying, gatewaying and other advanced functionality.

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 (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 which case UUIDs should be used), 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).

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
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

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 *TODO: Add support for SAML 2?*

Detection

This section should be optional, or removed if there is a risk of it becoming a cruft repository/interoperability nightmare.

Clients can detect the presence of the interface from the presence of a specific well-known URI at the web server root (as detected via a successful HTTP HEAD or GET):

```
http://example.com/.well-known/occi/
```

Tip

This may be achieved simply by creating the relevant directory in a static web root (e.g. `/var/www/.well-known/occi`).

This URI *should* be accessible over HTTP for performance reasons but if a HTTP server is present then this URI *must* also be accessible over HTTP. Clients *may* fall back to HTTPS if the HTTP port is closed or use HTTPS exclusively or not at all.

Tip

If the server responds over HTTP but the request is unsuccessful then the client will understand that the interface is not present. If it does not respond then the client may retry over HTTPS.

The URI *may* require authentication. If it does the client *may* choose to authenticate or not (in which case the result of the test will be inconclusive).

Configuration

Clients *should* retrieve configuration metadata (e.g. icons, logos, SLAs, supported kinds, etc.) from the following well-known URI:

```
http://example.com/.well-known/occi/feeds
```

The configuration information includes:

- URIs to collections of supported kinds
- Provider information (name, contacts, icon, logo)

The response returned will contain groups of collectionsbe dependent on the negotiated rendering.

Tip

This may be one or more static documents served using a separate technology from the interface itself (e.g. Apache multiviews).

TODO: Detection of category information.

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 action 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, in the form of key-value pairs.

Attributes are divided into namespaces which are separated by the dash character (“-”). They *must* be handled as case-insensitive but *should* be case-preserving by default (depending on the format).

Tip

This scalable approach was derived from the Mozilla Firefox `about:config` page, though the “.” separator was replaced with “-” for maximum compatibility with various formats.

Attributes defined by this standard reside under the `OCCI` namespace (e.g. "`OCCI-X`") but anyone can define a new attribute by allocating a unique namespace (e.g. “Acme”). A number of attributes are common to all *kind*s.

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.

```
OCCI-Compute-Cores: 2
OCCI-Compute-Speed: 3000
OCCI-Memory-Size: 8192
Acme-Network-Identifier: dmz
```

Table 1. Common Attributes

Attribute	Description	Example
`OCCI-Id`	Immutable identifier for the resource	`urn:uuid:d0e9f0d0-f62d-4f28-bc90-23b0bd871770`
`OCCI-Title`	Display name for the resource	`Compute Resource #123`
`OCCI-Summary`	Description of the resource	`A virtual compute resource`
`OCCI-Author-Name`	Owner of the resource	`John Citizen`
`OCCI-Updated`	Last updated date/time [RFC3339]	`2020-12-31T23:59:59Z`
`ETag`	HTTP Entity Tags [RFC2616]	`"dad86c61eea237932f"`

Actions

An *action* is some process that can be carried out on one or more *resource*s. Each available *action* for a given *resource* is indicated via a *link* with the `action` class.

```
<link rel="http://purl.org/occi/action/restart#cold"
      class="action"
```

```
    title="Cold Restart"
    href="http://example.com/123/restart;type=cold" />
```

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-encoded`). 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`.

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.

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. “`http://purl.org/occi/kind/`”)
label (optional)	A human-friendly display name for the term (e.g. “`Compute Resource`”)

Warning

Categories provide an ultimately flexible way to manage resources by taxonomy (categories) and/or folksonomy (tags), where both can be shared between [groups of] users or globally.

```
<category term="compute"
          scheme="http://purl.org/occi/kind/"
```

```
                  label="Compute Resource" />
```

Collections

Where an operation returns multiple resources (e.g. categories, searches) this is referred to as a *collection*. Depending on the format these are returned as:

- A list of pointers to resources (e.g. `text/uri-list [http://tools.ietf.org/html/rfc2483#section-5]`)
- A list of pointers to resources with metadata (e.g. `application/atom+xml` with `link` to content)
- A list of embedded resources and metadata (e.g. `application/atom+xml` with `content` embedded)

Tip

Most collections should be pointers to resources with metadata for performance reasons - O(1) rather than O(n+1) for pointers alone. The resources themselves should only be embedded when they are known to be of a reasonable size.

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

Tip

The root (“/”) *should* expose collections in order for clients to discover resources.

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

Paging

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

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

Linking

Existing linking standards defined for Atom [RFC4287], 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 rel="related"
```

```
    title="System Documentation"
    href="http://example.com/user-guide.pdf"
    type="application/pdf" />
```

Table 2. 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]

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.

Registration

IANA Considerations

Internet Media Types (MIME Types)

The following media types are to be registered:

- text/occi

- application/occi+atom
- application/occi+json

Well-Known URI Registry

The following well-known URI suffix is to be registered:

URI Suffix	/.well-known/*occi*/
Change Controller	OGF
Specification Document	Open Cloud Computing Interface (OCCI) [`http://purl.org/occi`]
Related Information	N/A

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 as defined by RFC2045 [`http://tools.ietf.org/html/rfc2045`] and RFC2046 [`http://tools.ietf.org/html/rfc2046`]

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.