Draft
OCCI-WG

Andy Edmons, ICCLab, ZHAW
Thijs Metsch, Intel
January 28, 2016

Open Cloud Computing Interface – Text Rendering

Abstract

This document, part of a document series produced by the OCCI working group within the Open Grid Forum (OGF), provides a high-level definition of a Protocol and API. The document is based upon previously gathered requirements and focuses on the scope of important capabilities required to support modern service offerings.

Contents

1 Introduction

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

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

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

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

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

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

2 Notational Conventions

All these parts and the information within are mandatory for implementors (unless otherwise specified). The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [**?**].

[1] Infrastructure as a Service

3 Text rendering

This document presents the text-based renderings. To be complaint, OCCI implementations MUST implement the three renderings defined in sections 6, 7 and 8.

The following specification of the text-based renderings is in the process of being deprecated and will be removed or significantly changed in the next MAJOR release of the standard.

The document is structured by defining base ABNFs, which can then be combined into renderings, which will be rendered over a protocol (e.g., HTTP) by the specific rendering definitions.

4 ABNF Definitions

For the following section of 5 these ABNF notations will be used. Implementations MUST hence implement the renderings according to these definitions.

4.1 Category ABNF

The following syntax MUST be used for Category renderings:

```
Category              = "Category" ":" #category-value
  category-value      = term
                         ";" "scheme" "=" <"> scheme <">
                         ";" "class" "=" ( class | <"> class <"> )
                         [ ";" "title" "=" quoted-string ]
                         [ ";" "rel" "=" <"> type-identifier <"> ]
                         [ ";" "location" "=" <"> URI <"> ]
                         [ ";" "attributes" "=" <"> attribute-list <"> ]
                         [ ";" "actions" "=" <"> action-list <"> ]
  term                = ( ALPHA | DIGIT ) *( ALPHA | DIGIT | "-" | "_" | "." )
  scheme              = URI
  type-identifier     = scheme term
  class               = "action" | "mixin" | "kind"
  attribute-list      = attribute-def
                      | attribute-def *( 1*SP attribute-def)
  attribute-def       = attribute-name
                      | attribute-name
                        "{" attribute-property *( 1*SP attribute-property ) "}"
  attribute-property  = "immutable" | "required"
  attribute-name      = term
  action-list         = action
                      | action *( 1*SP action )
  action              = type-identifier
```

4.2 Link ABNF

The following syntax MUST be used to represent OCCI Link type instance references:

```
Link                = "Link" ":" #link-value
  link-value        = "<" URI-reference ">"
                       ";" "rel" "=" <"> resource-type <">
                       [ ";" "self" "=" <"> link-instance <"> ]
                       [ ";" "category" "=" link-type
                         *( ";" link-attribute ) ]
```

```
  term             = ( ALPHA | DIGIT ) *( ALPHA | DIGIT | "-" | "_" | "." )
  scheme           = URI
  type-identifier  = scheme term
  resource-type    = type-identifier *( 1*SP type-identifier )
  link-type        = type-identifier *( 1*SP type-identifier )
  link-instance    = URI-reference
  link-attribute   = attribute-name "=" ( token | quoted-string )
  attribute-name   = term
```

The following syntax MUST be used to represent OCCI Action instance references:

```
ActionLink         = "Link" ":" #link-value
  link-value       = "<" action-uri ">"
                      ";" "rel" "=" <"> action-type <">
  term             = ( ALPHA | DIGIT ) *( ALPHA | DIGIT | "-" | "_" | "." )
  scheme           = URI
  type-identifier  = scheme term
  action-type      = type-identifier
  action-uri       = URI "?" "action=" term
```

4.3 Attribute ABNF

```
Attribute          = "X-OCCI-Attribute" ":" #attribute-repr
  attribute-repr   = attribute-name "=" attribute-value
  attribute-name   = ( ALPHA | DIGIT ) *( ALPHA | DIGIT | "-" | "_" | "." )
  attribute-value  = ( string | number | bool | enum-val )
  string           = quoted-string
  number           = (int | float)
  int              = *DIGIT
  float            = *DIGIT "." *DIGIT
  bool             = ("true" | "false")
  enum-val         = string
```

4.4 Location ABNF

```
Location      = "X-OCCI-Location" ":" location-value
  location-value  = URI-reference
```

5 Renderings

The renderings defined in this section will be used in the specific text rendering defined in section 6 and 7

5.1 Entity Instance Rendering

Entity instances MUST be rendered according to the following definitions.

5.1.1 Resource Instance Rendering

A Resource instance MUST be rendered using the following definition:

```
  resource_rendering = 1*( Category CRLF )
                       *( Link CRLF )
                       *( Attribute CRLF )
```

The rendering of a Resource instance MUST represent any associated Action instances using the `ActionLink CRLF`.

5.1.1.1 Action Invocation Rendering

Upon an Action invocation the client MUST send along the following definition:

```
action_definition = 1( Category CRLF )
                    *( Attribute CRLF )
```

5.1.2 Link Instance Rendering

A Link instance MUST be rendered using the following definition:

```
link_rendering = 1*( Category CRLF )
                 *( ActionLink CRLF )
                 *( Attribute CRLF )
```

5.2 Category Instance Rendering

A Category instances MUST be rendered as defined below.

5.2.1 Kind Instance Rendering

A Kind instance MUST be rendered as a `Category CRLF`.

5.2.2 Mixin Instance Rendering

A Mixin instance MUST be rendered as a `Category CRLF`.

5.2.3 Action Instance Rendering

An Action instance MUST be rendered as a `Category CRLF`.

Note that an Action instance MUST NOT have Link and Actions references.

5.3 Entity Collection Rendering

A collection of Resource or Link instances MUST be rendered as following:

```
entity_collection_rendering = *( Location CRLF )
```

5.3.1 Resource Collection Rendering

see above

5.3.2 Link Collection Rendering

see above

5.4 Category Collection Rendering

For the Query interface the following Category instance rendering MUST be used:

```
category_collection_rendering = *( Category CRLF )
```

5.4.1 Kind Collection Rendering

see above

5.4.2 Mixin Collection Rendering

see above

5.4.3 Action Collection Rendering

see above

5.5 Attributes Rendering

5.5.1 Entity Instance Attribute Rendering Specifics

For Entity instances the following model attribute name to attribute name rendering mappings MUST be used:

Table 2. Entity attribute naming convention

Attribute	Attribute name once rendered
Entity.id	occi.core.id
Entity.title	occi.core.title
Resource.summary	occi.core.summary
Link.target	occi.core.target
Link.target.kind	occi.core.target.kind
Link.source	occi.core.source
Link.source.kind	occi.core.source.kind

5.5.2 Mixin Instance Attribute Rendering Specifics

When rendering `Mixin.depends` and `Mixin.applies` to the `rel` attribute in the Category instance rendering, only `Mixin.depends` value MUST be used. If `Mixin.depends` contains multiple values, only the first value MUST be used.

5.5.3 Attribute Description Rendering

Attributes MUST be rendered as defined by the `Attribute CRLF`. If used, the `pattern` model attribute MUST be represented as a string in the ERE [**?**] format.

6 OCCI Text Plain rendering

The OCCI Text plain rendering specifies a rendering of OCCI instance types in a simple text format.

The rendering can be used to render OCCI instances independently of the protocol being used. Thus messages can be delivered by, e.g., the HTTP protocol as specified in [**?**].

The following media-types MUST be used for the OCCI Text plain rendering:

`text/occi+plain`

and

`text/plain`

Each entry in the body consists of a name followed by a colon (":") and the field value.

6.1 Example

The following example show an Entity instance rendering using the Text plain rendering.

```
< Category: compute; \
<     scheme="http://schemas.ogf.org/occi/infrastructure#" \
<     class="kind";
< Link: </users/foo/compute/b9ff813e-fee5-4a9d-b839-673f39746096?action=start>; \
<     rel="http://schemas.ogf.org/occi/infrastructure/compute/action#start"
< X-OCCI-Attribute: occi.core.id="urn:uuid:b9ff813e-fee5-4a9d-b839-673f39746096"
< X-OCCI-Attribute: occi.core.title="My Dummy VM"
< X-OCCI-Attribute: occi.compute.architecture="x86"
< X-OCCI-Attribute: occi.compute.state="inactive"
< X-OCCI-Attribute: occi.compute.speed=1.33
< X-OCCI-Attribute: occi.compute.memory=2.0
< X-OCCI-Attribute: occi.compute.cores=2
< X-OCCI-Attribute: occi.compute.hostname="dummy"
```

7 OCCI Header Rendering

The following media-type MUST be used for the OCCI header Rendering:

`text/occi`

While using this rendering the HTTP Protocol [**?**] MUST be used and the renderings 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 7230 [**?**]. A header field consists of a name followed by a colon (":") and the field value.

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 7230 MUST be fully supported. In essence there are two valid representations 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 7230.

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 implementations and take appropriate measures so that truncation of header data does NOT occur.

7.1 Example

The following example shows an Entity instance rendering using the Text header rendering.

```
< Category: compute; \
    scheme="http://schemas.ogf.org/occi/infrastructure#" \
    class="kind";
< Link: </users/foo/compute/b9ff813e-fee5-4a9d-b839-673f39746096?action=start>; \
    rel="http://schemas.ogf.org/occi/infrastructure/compute/action#start"
< X-OCCI-Attribute: occi.core.id="urn:uuid:b9ff813e-fee5-4a9d-b839-673f39746096", \
 occi.core.title="My Dummy VM", occi.compute.architecture="x86", \
 occi.compute.state="inactive", occi.compute.speed=1.33, \
 occi.compute.memory=2.0, occi.compute.cores=2, \
 occi.compute.hostname="dummy"
< OK
```

8 URI Listing Rendering

The following media-types MUST be used for the URI Rendering:

`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. Otherwise a list of resources MUST be rendered in `text/uri-list` format as defined in [**?**], which can be used for listing resource in collections or the name-space of the OCCI implementation.

9 Security Considerations

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 MAY be used with OCCI are those that can be used with HTTP and TLS. For further discussion see the appropriate section in [**?**].

10 Glossary

Term	Description
Action	An OCCI base type. Represents an invocable operation on an Entity sub-type instance or collection thereof.
Attribute	A type in the OCCI Core Model. Describes the name and properties of attributes found in Entity types.
Category	A type in the OCCI Core Model and the basis of the OCCI type identification mechanism. The parent type of Kind.
capabilities	In the context of Entity sub-types **capabilities** refer to the Attributes and Actions exposed by an **entity instance**.
Collection	A set of Entity sub-type instances all associated to a particular Kind or Mixin instance.
Entity	An OCCI base type. The parent type of Resource and Link.
entity instance	An instance of a sub-type of Entity but not an instance of the Entity type itself. The OCCI model defines two sub-types of Entity: the Resource type and the Link type. However, the term *entity instance* is defined to include any instance of a sub-type of Resource or Link as well.
Kind	A type in the OCCI Core Model. A core component of the OCCI classification system.
Link	An OCCI base type. A Link instance associates one Resource instance with another.
Mixin	A type in the OCCI Core Model. A core component of the OCCI classification system.
mix-in	An instance of the Mixin type associated with an *entity instance*. The "mix-in" concept as used by OCCI *only* applies to instances, never to Entity types.
OCCI	Open Cloud Computing Interface.
OGF	Open Grid Forum.
Resource	An OCCI base type. The parent type for all domain-specific Resource sub-types.
resource instance	See *entity instance*. This term is considered obsolete.
tag	A Mixin instance with no attributes or actions defined. Used for taxonomic organisation of entity instances.
template	A Mixin instance which if associated at instance creation-time pre-populate certain attributes.
type	One of the types defined by the OCCI Core Model. The Core Model types are Category, Attribute, Kind, Mixin, Action, Entity, Resource and Link.
concrete type/sub-type	A concrete type/sub-type is a type that can be instantiated.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
URN	Uniform Resource Name.

11 Contributors

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

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

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

12 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.

13 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.

14 Full Copyright Notice

A Change Log

The corrections introduced by the January 28, 2016 update are summarized below. This section describes the possible impact of the corrections on existing implementations and associated dependent specifications.

- Relaxed rules on `term` values allowing the use of: alphanumerical characters (a-zA-Z0-9), "_", "-" and ".".
- Explicitly stated how `Mixin.depends` and `Mixin.applies` should be rendered to `rel` on Mixin instances.