Draft
OCCI-WG

Ralf Nyrén, Independent
Andy Edmonds, ICCLab, ZHAW
Alexander Papaspyrou, Adesso
Thijs Metsch, Intel
Boris Parák, CESNET
Update: March 19, 2015

Open Cloud Computing Interface - Core

Status of this Document

This document is a draft including proposed errata updates to the OCCI Core [1] specification.

The errata updates are summarized in section A.

Eventually this document will obsolete GFD-P-R.183. This document is fully backward compatible to [1].

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 complimentary 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 through *renderings* (including associated behaviours) 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 [2].

[1] Infrastructure as a Service

3 Terms and definitions

Section 7 provides a glossary of all terms and definitions with a specific meaning to the OCCI specification suite. However, for reader convenience, a sub-set of the glossary is provided here as well. The following terminology has specific meaning in the OCCI context:

capabilities In the context of Entity sub-types **capabilities** refer to the Attributes and Actions exposed by a **entity instance**.

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.

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. See section 5.3.4.

model attribute An attribute of a the Core Model.

OCCI base type(s) The OCCI base types are Entity, Resource and Link. See section 5.4.

template A mechanism to provide default values for a **entity instance**. See section 5.3.7.

type A **type** refer to one of those defined by the OCCI Core Model. The OCCI Core Model types are Category, Attribute, Kind, Mixin, Action, Entity, Resource and Link.

concrete type/sub-type A concrete sub-type is a type that can be instantiated.

4 OCCI Core

The Open Cloud Computing Interface is a boundary protocol and API that acts as a service front-end to a provider's internal management framework. Figure 1 shows OCCI's place in a provider's architecture.

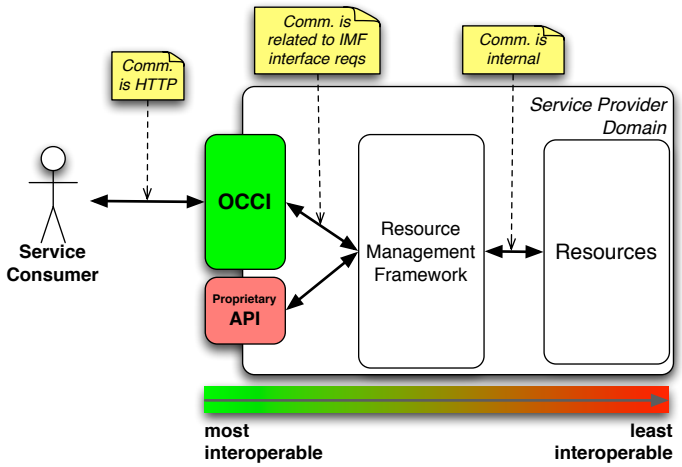

Figure 1. OCCI's place in a provider's architecture.

Service consumers can be both end-users and other system instances. OCCI is suitable for both cases. The key feature is that OCCI can be used as a management API for all kinds of resources while at the same time maintaining a high level of interoperability.

This document, the OCCI Core specification, defines the OCCI Core Model. This model is the core of the specification suite and it can be interacted with by renderings (including associated behaviours) and expanded through extensions. In itself, the core model is only useful for a very limited set of use cases. However, it provides the basis for renderings and extensions to build upon.

5 OCCI Core Model

The OCCI Core Model defines a representation of instance types which can be manipulated through an OCCI protocol and rendering implementations. It is an abstraction of real-world resources, including the means to identify, classify, associate and extend those resources.

A fundamental feature of the OCCI Core Model is that it can be extended in such a way that any extension will be discoverable and visible to an OCCI client at run-time. An OCCI client can connect to an OCCI implementation using an extended OCCI Core Model, without knowing anything in advance, and still be able to discover and understand, at run-time, the various instance types supported by that implementation. For example, a web-based OCCI client could easily be reused as the management tool for a wide variety of services.

The OCCI Core Model can be extended through inheritance but also using a "mix-in" like concept.

> Mixins first appeared in the Symbolics' object-oriented Flavors [3] system (developed by Howard Cannon), which was an approach to object-orientation used in Lisp Machine Lisp.[2]

The mix-in model only applies at the instance level, i.e. the "object level", and thereby differs from the more common uses of the mix-in concept. A mix-in in OCCI can never be applied to a type, only to an instance.

5.1 Overview

The UML class diagram shown in figure 2 gives an overview of the OCCI Core Model. It must be noted that the UML diagram in itself is not a complete definition of the model. The diagram is merely provided as an overview to help understanding the model.

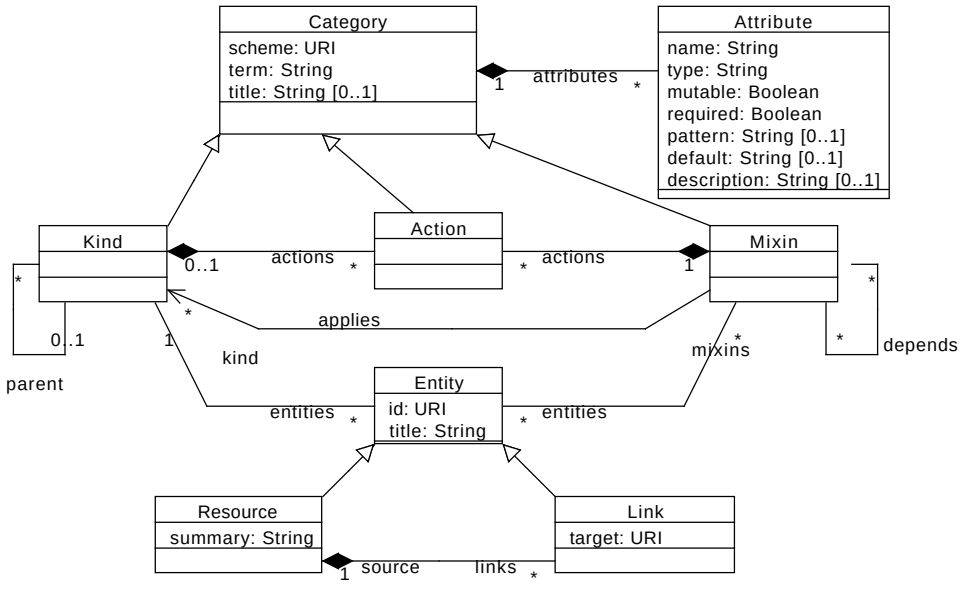

Figure 2. UML class diagram of the OCCI Core Model. The diagram provides an overview of the OCCI Core Model but is not a standalone definition thereof.

The heart of the OCCI Core Model is the Resource type. Any resource exposed through OCCI is a Resource or a sub-type thereof. A resource can be e.g. a virtual machine, a job in a job submission system, a user, etc.

The Resource type contains a number of common attributes that Resource sub-types inherit. The Resource type is complemented by the Link type which associates one Resource instance with another. The Link type contains a number of common attributes that Link sub-types inherit.

[2] http://en.wikipedia.org/wiki/Mixin.

Entity is an abstract type, which both Resource and Link inherit. Each sub-type of Entity is identified by a unique Kind instance.

The Kind type is the core of the type classification system built into the OCCI Core Model. Kind is a specialisation of Category and introduces additional capabilities in terms of Actions. An Action identifies an invocable operation applicable to an entity instance.

Attribute describe the name and properties of the Attributes found in Entity and its sub-types.

The last type defined by the OCCI Core Model is the Mixin type. An instance of Mixin can be associated with an entity instance to "mix-in" additional capabilities at run-time.

For compliance with OCCI Core, all of the types defined in the OCCI Core Model MUST be implemented. The following sections of the specification contain the formal definition of the OCCI Core Model.

5.2 Mutability

Attributes of an OCCI Core Model type instance are either client mutable or client immutable. If an attribute is noted to be mutable this MUST be interpreted that a client can create an instance that is parametrised by the attribute. Likewise, if an attribute is mutable, a client can update that instance's mutable attribute value and the server side MUST support this. If an attribute is marked as immutable, it indicates that the server side implementation MUST manage these exclusively. Immutable attributes MUST NOT be modifiable by clients under any circumstance.

5.3 Classification and Identification

The OCCI Core Model provides a built-in type classification system allowing for safe extension towards domain-specific usage (e.g. infrastructure). This system is the OCCI type system and offers the means to be easily and transparently (i.e. no format translation required) exposed over either a text- or binary-based protocol.

The classification system can be summarised with the following key features:

- Each OCCI base type and extension thereof is assigned a unique type identifier (a Kind instance), which allow for dynamic discovery of available types. All Entity sub-types, including core model extensions, are assigned a unique Kind instance.
- The inheritance structure of Entity, Resource and Link is client discoverable. This also applies to any sub-type of Resource and Link and therefore an OCCI client can discover the type inheritance structure used by a particular OCCI implementation. The discovery of the inheritance structure is made possible through the relationship of Kind instances.
- The classification system allows Mixin instances to be associated to entity instances in order to assign additional capabilities in terms of Attributes and Actions at run-time.
- Tagging of entity instances is supported through the association of Mixin instances. A tag is simply a Mixin instance, which defines no additional capabilities.
- A collection of associated entity instances is implicitly defined for each Kind and Mixin instance. That is, all entity instances associated with a particular Kind or Mixin instance form a collection.

5.3.1 Category

The Category type is the basis of the type identification mechanism used by the OCCI classification system. It MUST be implemented. There are no instances of the Category type itself in the OCCI Core Model. The Category type is only used through its sub-types Kind, Mixin and Action. Table 2 defines the model attributes the Category type MUST implement to be compliant.

Table 2. Model attributes defined for the Category type.

Model attribute	Type	Multiplicity	Client Mutability	Description
term	String	1	Immutable	Unique identifier of the Category instance within the categorisation scheme.
scheme	URI	1	Immutable	The categorisation scheme.
title	String	0..1	Immutable	The display name of an instance.

A Category instance is uniquely identified by concatenating the categorisation scheme with the category term, e.g. *http://example.com/category/scheme#term*. This is done to enable discovery of Category definitions in text-based renderings such as HTTP. All renderings MUST make use of and understand concatenated unique type identifiers of Category instances. Sub-types of Category such as Kind, Mixin and Action inherit this property.

The categorisation schemes defined in the OCCI specification all use the *http://schemas.ogf.org/occi/* base URL. This base URL is reserved for OCCI an MUST NOT be used by service provider extensions.

A Category instance[3] have zero or more associated Attribute instances. Each Attribute, see section 5.3.2, describes the name and properties of single attribute.

5.3.2 Attribute

The Attribute type has a composite relationship to Category and defines the name and properties of client readable Attributes. Table 3 defines the model attributes the Attribute type MUST implement to be compliant.

Table 3. Model attributes defined for the Attribute type.

Model attribute	Type	Multiplicity	Client Mutability	Description
name	String	1	Immutable	Attribute name.
type	Enum {Object, List, Hash}	1	Immutable	Attribute type.
mutable	Boolean	1	Immutable	Attribute mutability.
required	Boolean	1	Immutable	Whether the Attribute must be supplied by the client at instance creation-time.
pattern	String	0..1	Immutable	Attribute pattern expressed as ERE
default	String	0..1	Immutable	Attribute default value.
description	String	0..1	Immutable	A description of the Attribute.

An Attribute name MUST be defined by Attribute.`name`. The Attribute namespace is flat and the "`occi.`" prefix is reserved for the OCCI specification. Domain-specific Attribute names MUST NOT contain the "`occi.`" prefix, instead they SHOULD use a prefix consisting of the provider's reverse domain name. E.g. "`com.example.`".

An Attribute MAY specify the following properties in addition to the Attribute name. Attribute properties are OPTIONAL but MUST be client discoverable if used.

type The type of the Attribute. The types supported are "Object", "List" and "Hash".

mutable Whether an OCCI client can change the Attribute value. See section 5.2.

required If an Attribute is "required" a client MUST specify an value at instance creation-time.

pattern MAY be specified in ERE [4] format, places additional restrictions on possibles values given.

default The default value given to an Attribute if the client does not specify a value at instance creation-time. The *default* property is used to implement templates, see section 5.3.7.

description A summarizing description of the Attribute to complement the attribute name. For example, an interactive OCCI client may use the description property when presenting the content of an entity instance.

[3] Also applies to Kind, Mixin and Action instances.

5.3.3 Kind

The Kind type, together with the Mixin type, defines the classification system of the OCCI Core Model. It MUST be implemented. The Kind type represents the type identification mechanism for all Entity types present in the model. Sub-types MUST NOT be derived from the Kind type.

A unique Kind *instance* MUST be assigned to each and every Entity sub-type defined in an OCCI implementation.

Every instance of Kind represents a unique type identifier for a particular sub-*type* of Entity. Consequently, when an Entity sub-type is instantiated the entity instance MUST be associated with its type identifier, i.e. the Kind instance. An entity instance MUST remain associated with its Kind instance throughout its lifetime. For example an instance of Resource MUST always be associated with the Kind instance which identifies the Resource *type*.

In the initial instantiation of the OCCI Core Model, with no core model extensions, three instances of Kind will be present: one for Entity, another for Resource and the last one for Link.

Table 4. Model attributes defined for the Kind type.

Model attribute	Type	Multiplicity	Client Mutability	Description
actions	Action	0..*	Immutable	Set of Action instances defined by the Kind instance.
parent	Kind	0..1	Immutable	Another Kind instance which this Kind has an inheritance relationship with.
entities	Entity	0..*	Immutable	Set of entity instances. Instances of the particular Entity sub-type which is uniquely identified by this Kind instance.

The Kind type inherits the Category type. To be compliant the Kind type MUST implement the model attributes defined in table 4 and the inherited model attributes defined in table 2. The following rules apply to all instances of the Kind type:

- A unique Kind instance MUST be assigned to each and every sub-type of Entity, including Entity itself.
- A Kind instance MUST expose the discoverable attributes defined for the Entity sub-type it identifies.
- A Kind instance MUST expose the Actions defined for its Entity sub-type.
- A Kind instance MUST have the Kind instance of Entity[4] as its parent.
- If type **B** inherits type **A**, where **A** is a sub-type of Entity, the Kind instance of **B** MUST have its `parent` attribute set to the Kind instance of **A**. See Kind Relationships below.

Kind Relationships A relationship between Kind instances is defined by the "`parent`" attribute. This implies a setup of a hierarchy where the capabilites of the parent MUST be inherited by the child Kind instance.

Figure 3 illustrates the relationship of the Kind instances assigned to the Entity, Resource and Compute[5] types. Compute inherits Resource and therefore the Kind instance assigned to Compute has the Kind instance of Resource as its parent. The same applies to the Resource type which inherit Entity.

As can be seen in figure 3 the Kind instance relationships mirror the inheritance structure of the types.

5.3.4 Mixin

The Mixin type complements the Kind type in defining the OCCI Core Model type classification system. It MUST be implemented. The Mixin type represent an extension mechanism, which allows new capabilities to be added to entity instances both at creation-time and/or run-time. Sub-types MUST NOT be derived from the Mixin type.

[4] *http://schemas.ogf.org/occi/core#entity*
[5] The Compute type is defined in the OCCI Infrastructure document [5].

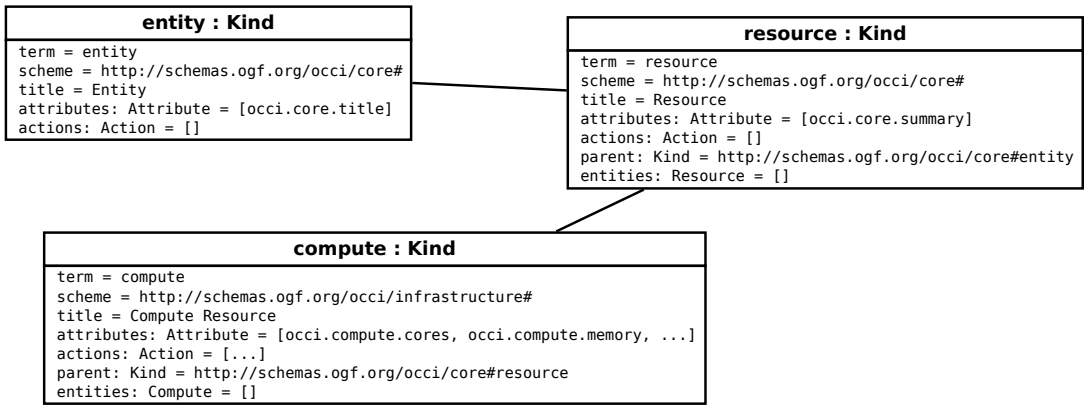

Figure 3. Object diagram illustrating the Kind instances involved for the Entity, Resource and Compute types. The Compute type is an extension to the OCCI Core Model defined in the OCCI Infrastructure document [5].

A Mixin *instance* can be associated with any existing entity instance and thereby identify new capabilities, i.e. Attributes and Actions, for the entity instance. However, a Mixin can never be applied to a type. In the initial instantiation of the OCCI Core Model, with no extensions, no Mixin instances are present.

A Mixin instance MAY be associated with an entity instance either at instance creation-time or at run-time. Restrictions on which entity instances a particular Mixin can be associated to SHOULD be advertised through the Mixin.`applies` model attribute.

When a client attempts to associate a Mixin instance to an entity instance at a stage not supported by a particular provider's OCCI implementation, the provider MUST notify the client it has issued a bad request. For example a "bandwidth" Mixin may only be applicable to instances of the Network[6] type. An OCCI provider SHOULD advertise such a restriction by setting Mixin.`applies` to the Kind instance of the Network type[7].

Table 5. Model attributes defined for the Mixin type.

Model attribute	Type	Multiplicity	Client Mutability	Description
actions	Action	0..*	Immutable	Set of Action instances defined by the Mixin instance.
depends	Mixin	0..*	Immutable	Set of Mixin instances this Mixin instance depends on.
applies	Kind	0..*	Immutable	Set of Kind instances this Mixin instance applies to.
entities	Entity	0..*	Mutable	Set of entity instances associated with the Mixin instance.

The Mixin type inherits the Category type. To be compliant the Mixin type MUST implement the model attributes defined in table 5 and the inherited model attributes defined in table 2. The following rules apply to all instances of the Mixin type:

- A Mixin instance MUST only be associated with entity *instances*, not types, either at creation-time or run-time.
- A Mixin instance is *only* a type identifier. It MUST NOT provide the implementation of the new capabilities it introduces. For example, a Mixin instance never contains the value of an OCCI Attribute.
- A Mixin instance MAY introduce additional Attributes when applied to an entity instance. The name and properties of those Attributes MUST be exposed through Mixin.`attributes` inherited from Category. E.g. a Location Mixin defining the "com.example.location" Attribute MUST have Location.`attributes` populated with a single Attribute instance where Attribute.`name` is `"com.example.location"`.
- A Mixin instance MAY define Action instances that will identify additional invocable operations on any entity instance associated with the Mixin. Actions defined by a Mixin are exposed through the Mixin.`actions` model attribute that represent the association between a Mixin instance and the Action instances it defines.

[6] The Network type is defined in OCCI Infrastructure [5].
[7] *http://schemas.ogf.org/occi/infrastructure#network*

- A Mixin instance MAY depend on another Mixin instance. If Mixin **B** depends on Mixin **A**, any entity instance associated with Mixin **B** will receive the capabilities defined by both Mixin **B** and Mixin **A**. See Mixin Relationships below.
- A Mixin instance defining no additional capabilities is considered to be a tag.
- A Mixin instance MAY be used as a template. A template defines default values for Attributes to be applied at entity instance creation-time. See section 5.3.7.
- A Mixin instance MAY restrict which Kind instances it applies to using the `applies` model attribute. If Mixin.`applies` is unspecified the Mixin may be associated to any entity instance, i.e. equivalent of having Mixin.`applies` set to the Kind instance of Entity.

Mixin Relationships A Mixin instance MAY be depend on other Mixin instances. Mixin relationships are implemented using the Mixin.`depends` model attribute. For example a set of operating system templates, implemented as Mixin instances, could be related to an "OS-template" Mixin in order to help identification.

Attributes and Actions defined by different Mixin instances are *combined* when Mixin relationships are present. Therefore an entity instance associated with a particular Mixin will receive the additional capabilities defined by any related Mixin instances as well as those defined by the Mixin associated.

5.3.5 Action

The Action type is the final part of the OCCI type classification system and identifies invocable operations on individual entity instances and collections. It MUST be implemented. Each Action instance identifies a single invocable operation. The Action instance is only an identifier and does not represent the implementation of the operation.

The Action type inherit the Category type. To be compliant the Action type MUST implement the inherited model attributes defined in table 2.

Table 6. Example of an Action instance which identifies a "resize" operation.

Model attribute	Value
term	resize
scheme	`http://schemas.ogf.org/occi/infrastructure/storage/action#`
title	Resize virtual disk
attributes	Attribute("`resize`")

An Action instance MUST always bound to either a Kind or a Mixin instance through a composite association. An Action is considered to be a capability of the Kind or Mixin instance it is associated with. The operation identified by an Action MAY be invoked on any entity instance associated with the Kind or Mixin instance defining the Action. An OCCI implementation MAY however refuse an the operation from being invoked if currently not applicable.

The operation identified by an Action instance MAY be invoked on a collection of Entity sub-type instances. The Action is only considered valid if all entity instances of the collection are associated with the Kind or Mixin defining the Action instance.

An Action instance MAY identify Attributes which correspond to parameters of the invocable operation. The mechanism to define Attributes is inherited from Category and follow the same semantics. The namespace restrictions imposed on entity instance attributes (see 5.3.2) does however not apply to Actions.

Table 6 shows an example of a "resize" operation defined for a Storage instance. The operation has a "size" parameter which represent the size argument of the resize operation. In that example the identifying Action instance would have Action.`attributes` populated with an Attribute instance where Attribute.`name` = "`size`".

5.3.6 Instantiation

To create an entity instance a client MUST supply the concrete Entity sub-type by a submitting a reference to the type-identifying Kind. The reference MUST consist of the term and categorisation scheme which uniquely identify the Kind instance, see section 5.3.1. All OCCI implementations MUST understand these requests.

A client MAY also submit any number of references to Mixin instances to be associated with the instance to be created. A Mixin reference submitted by a client MUST consist of the term and categorisation scheme which identify the Mixin instance, see section 5.3.1.

5.3.7 Templates

A template is a mechanism to provide default values for entity instances. OCCI supports templates through Mixins.

A Mixin instance associated at entity instance creation-time MAY provide default values for Attributes. Each default value is specified through Attribute.`default`.

A Mixin instance MAY provide default values for Attributes already defined by a Kind. A Mixin's Attribute.`default` overrides the default specified by the Kind.

5.3.8 Collections

One or more entity instances associated with the same Kind or Mixin instance, automatically form a collection. Each Kind and Mixin instance in the system identifies a collection consisting of all different entity instances associated with the same Kind or Mixin.

An entity instance is always a member of the collection indicated by the Entity sub-type's unique Kind instance. The Kind.`entities` model attribute implements the collection of entity instances for a specific Entity sub-type.

A Kind instance maintains the collection of all entity instances (of the type identified by the Kind).

Since a Mixin instance can be associated to any entity instance, a collection can contain entity instances of different Entity sub-types. For example, an instance of the Resource type will always be associated to the Kind instance *http://scheme.ogf.org/occi/core#resource* and thus part of the collection implied by that Kind instance.

Adding an entity instance to a collection is accomplished by associating the entity instance to the corresponding Mixin instance.

Removing an entity instance from a collection is accomplished by disassociating the entity instance from the corresponding Mixin instance.

An OCCI implementation MUST allow a client to navigate collections. The following basic navigation operations MUST be supported:

- Retrieve the whole collection.
- Retrieve a specific item in a collection.
- Retrieve a subset of a collection.

The details of collection navigation is rendering specific.

5.3.9 Discovery

An OCCI client MUST be able to discover all instances of Kind, Mixin and Category a particular service provider's OCCI implementation has defined. By examining these instances a client MUST be able to, at a minimum, deduce the following information:

- The Entity sub-types available from the service provider, including core model extensions. This information is provided through the Kind instances of the OCCI implementation.
- The attributes defined for each Entity sub-type. The identifying Kind instance provide this information.
- The invocable operations, i.e. Actions, defined for each Entity sub-type. The identifying Kind instance provide this information.
- Any Mixin instances that can be associated to entity instances.
- Additional capabilities defined by a particular Mixin instance, i.e. Attributes and Actions.

The above requirements comprise the OCCI discovery mechanism. It MUST be implemented.

The details of exactly how the Category, Kind and Mixin instances are exposed to an OCCI client is specific to the particular rendering used. The relevant details can be found in the OCCI Rendering documents.

5.4 The OCCI Core Base Types

The following sections describe the OCCI base types defined by the OCCI Core Model. The base types are Entity, Resource, Link. All base types MUST be implemented.

An instance of the Resource type, the Link type or one of their sub-types is called a *entity instance*. Each entity instance within an OCCI system MUST have a unique identifier[8] stored in the `id` model attribute of the Entity type, as defined in table 7. The structure of these identifiers is opaque and the system should not assume a static, pre-determined scheme for their structure other than the rules imposed by the Uniform Resource Identifier (URI) [6] syntax.

Although every unique entity instance identifier MUST be valid URI it is RECOMMENDED to use the Uniform Resource Name (URN) [7] syntax.

For example Entity.`id` could be `urn:uuid:de7335a7-07e0-4487-9cbd-ed51be7f2ce4`.

5.4.1 Entity

The Entity type is an abstract type of the Resource type and the Link type. It MUST be implemented. Table 7 defines the model attributes the Entity type MUST implement to be compliant.

Table 7. Model attributes defined for the Entity type.

Model attribute	Type	Multiplicity	Client Mutability	Description
id	URI	1	Immutable	A unique identifier (within the service provider's name-space) of the Entity sub-type instance.
title	String	0..1	Mutable	The display name of the instance.
kind	Kind	1	Immutable	The Kind instance uniquely identifying the particular Entity sub-type of this instance.
mixins	Kind	0..*	Mutable	The Mixin instances associated to this entity instance. Consumers can expect the Attributes and Actions of the associated Mixins to be exposed by the instance.

Every sub-type of Entity MUST be assigned a Kind instance, see section 5.3.3.

[8] An entity instance identifier MUST be unique within the service provider's name-space. It is RECOMMENDED to use globally unique identifiers.

Entity itself is assigned the Kind instance *http://schemas.ogf.org/occi/core#entity*. Being an abstract type Entity itself can never be instantiated.

An Entity sub-type instance, also referred to as an *entity instance*, MAY be associated with one or more Mixin instances.

An Entity sub-type instance MUST expose its identifying Kind instance and any associated Mixin instances together with the Attributes and Actions defined by them.

5.4.2 Resource

The Resource type inherits Entity and describes a concrete resource that can be inspected and manipulated. It represents a general object in the OCCI model and MUST be implemented. A Resource is suitable to represent real world resources, e.g. virtual machines, networks, services, etc. through specialisation.

Table 8. Model attributes defined for the Resource type.

Model attribute	Type	Multiplicity	Client Mutability	Description
links	Link	0..*	Mutable	A set of Link compositions. Being a composite relation the removal of a Link from the set MUST also remove the Link instance.
summary	String	0..1	Mutable	A summarising description of the Resource instance.

The Resource type is assigned the Kind instance *http://schemas.ogf.org/occi/core#resource*.

Resource enforces the inheritance of a set of common attributes into sub-types. Moreover, it introduces relationships to other Resource instances through instances of the Link type.

The Resource type is the first of three entry points to extend the OCCI Core Model, see section 5.5.

5.4.3 Link

An instance of the Link type defines a base association between two Resource instances. It MUST be implemented. A Link instance indicates that one Resource instance is connected to another.

The Link type MUST implement all attributes inherited from the Entity type together with the model attributes defined in table 9 in order to be compliant.

Table 9. Model attributes defined for the Link type.

Model attribute	Type	Multiplicity	Client Mutability	Description
source	Resource	1	Mutable	The Resource instances the Link instance originates from.
target	URI	1	Mutable	The unique identifier of an Object this Link instance points to.

The Link type is assigned the Kind instance *http://schemas.ogf.org/occi/core#link*.

The `source` attribute of a Link instance MUST refer to Resource *instance* within the service provider's namespace. The Link's `target` attribute MUST point to a resource instance either within the provider's namespace or outside, hosted by a third-party.

The Link type is the second of three entry points to extend the OCCI Core Model, see section 5.5.

5.5 Extensibility

The OCCI Core Model has a flexible yet fairly simple extension mechanism based on the type classification system described in section 5.3.

The OCCI Core Model can be extended using two different methods, sub-typing and mix-in. Custom sub-typing require provider-specific Kind instances and custom mix-ins require provider-specific Mixin instances. Both

methods MAY involve the use of provider-specific Action instances. The following sections define the rules for extending the OCCI Core Model.

The rules defined in section 5.3 and 5.4 are REQUIRED for all extensions of the OCCI Core Model.

5.5.1 Category instances

Provider-specific instances of Category, Kind and Mixin MAY be introduced by an OCCI implementation. Since Kind and Mixin both inherit Category the extension rules for Category, defined below, applies to them as well.

A Category instance defined outside of the OCCI specification MUST use a Category scheme unique to the provider, e.g. *http://example.com/occi#*. The `term` of a provider-specific Category instance can be any string corresponding to a "token" as defined by the OCCI Rendering documents.

An Attribute introduced by a provider-specific Category MUST use an attribute name prefix. This prefix MUST NOT be the "`occi.`" prefix which is reserved for the OCCI specification. Domain-specific Attribute names SHOULD use a prefix consisting of the provider's reverse domain name, e.g. "`com.example.`".

5.5.2 Sub-typing

The OCCI Core Model MAY be extended through sub-typing. Two OCCI Core Model types MAY be sub-typed, those are Resource and Link.

In order to define a new sub-type of Resource or Link, a provider-specific Kind instance MUST be defined and assigned to the new sub-type. This provider-specific Kind instance MUST have its Kind.`parent` model attribute equal to the Kind instance of the type extended. See figure 3 for an example of Kind relationships.

5.5.3 Mix-ins

The OCCI Core Model MAY be extended using a "mix-in" like concept by defining provider-specific Mixin instances. A Mixin instance can be associated with any entity instance although a provider MAY apply restrictions.

In order to support user-defined tags[9] an OCCI implementation must allow custom Mixin instances to be created and destroyed by request of a client. There is no limitation in the OCCI Core Model from doing so but it is RECOMMENDED to assign a separate Category scheme for each user's Mixin instances (e.g. per-user schemes).

6 Security Considerations

Since the OCCI Core and Model specification describes a model, not an interface or protocol, no specific security mechanisms are described as part of this document. However, the elements described by this specification, namely type instance attribute mutability, Category, Kind, and Mixin instantiations; Entity, Resource, and Link subtypes, whether direct or indirect; resource or collection manipulation; and the discovery mechanism need to implement a proper authorization scheme, which MUST be part of a concrete OCCI rendering specification, part of an OCCI specification profile, or part of the specific OCCI implementation.

Concrete security mechanisms and protection against attacks SHOULD be specified by OCCI rendering specification. In any case, OCCI rendering specifications MUST address transport level security and authentication on the protocol level.

All security considerations listed above apply to all (existing and future) extensions of the OCCI Core and Model specification.

[9] A tag is a Mixin instance, which does not introduce additional capabilities.

7 Glossary

Term	Description
Action	An OCCI base type. Represents an invocable operation on a 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.

8 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	NetApp	florian.feldhaus at gmail.com

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

9 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.

10 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.

11 Full Copyright Notice

References

[1] R. Nyrén, A. Edmonds, A. Papaspyrou, and T. Metsch, "Open Cloud Computing Interface – Core," GFD-P-R.183, April 2011. [Online]. Available: http://ogf.org/documents/GFD.183.pdf

[2] 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

[3] D. A. Moon, "Object-oriented programming with flavors," *SIGPLAN Not.*, vol. 21, pp. 1–8, June 1986. [Online]. Available: http://doi.acm.org/10.1145/960112.28698

[4] "Extened Regular Expressions," The Open Group, 1997. [Online]. Available: http://pubs.opengroup.org/onlinepubs/7908799/xbd/re.html

[5] T. Metsch and A. Edmonds, "Open Cloud Computing Interface – Infrastructure," GFD-P-R.184, April 2011. [Online]. Available: http://ogf.org/documents/GFD.184.pdf

[6] 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

[7] R. Moats, "URN Syntax," RFC 2141 (Proposed Standard), Internet Engineering Task Force, May 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2141.txt

[8] T. Metsch and A. Edmonds, "Open Cloud Computing Interface – HTTP Rendering," GFD-P-R.185, April 2011. [Online]. Available: http://ogf.org/documents/GFD.185.pdf

A Errata

The corrections introduced by the March 19, 2015 errata update are summarized below. The following subsections describe the possible impact of the corrections on existing implementations and associated dependent specifications such as OCCI HTTP Rendering [8] and OCCI Infrastructure [5].

- Introduce an explicit Attribute type to expose the discoverable attribute properties already defined for the OCCI base types Entity, Resource, Link and their sub-types.
- Correct the previously unclear definition of OCCI Action. The Action type inherits Category and is only an identifier of an invocable operation. It does *not* represent the operation itself. The Action definition now aligns with its use in the OCCI HTTP Rendering [8].
- Clarify the format of the unique entity instance identifier defined in OCCI Entity. Incorporate the description and recommendations from the OCCI HTTP Rendering [8].
- Clarify that an OCCI Mixin instance is only a type identifier. The Core Model does not specify how a mixed-in attribute is implemented. The Mixin instance only states that the attribute exists.
- Rename the term *resource instance* to *entity instance*. An *entity instance* refers to an instance of either OCCI Resource, OCCI Link or a sub-type of either type. The *resource instance* term, while defined identically, was due to its name a source of misinterpretations in the specification.
- Rename Kind.`related` to Kind.`parent` and Mixin.`related` to Mixin.`depends`. Clarify the use of Kind and Mixin relationships.
- Add a new model attribute Mixin.`applies` to optionally advertise which entity instances a Mixin instance may be associated to.

A.1 Action definition

The corrected definition of OCCI Action has no impact on neither discovery nor invocation of Actions in existing implementations. The OCCI HTTP Rendering [8] is better aligned with OCCI Core after the corrections since it already uses `type="action"` in its rendering of categories.

A.2 Rename “resource instance” to “entity instance”

The change is editorial and does not affect existing implementations. The glossary contains both terms for compatibility with the OCCI HTTP Rendering [8] specification.