

1 GFD-R-P.221
2 OCCI-WG

Ralf Nyrén, Independent
Andy Edmonds, ICCLab, ZHAW
Alexander Papaspyrou, Adesso
Thijs Metsch, Intel
Boris Parák, CESNET
Update: September 5, 2016

7 **Open Cloud Computing Interface – Core**

8 Status of this Document

9 This document provides information to the community regarding the specification of the Open Cloud Computing
10 Interface. Distribution is unlimited.

11 This document obsoletes GFD-P-R.183.

12 Copyright Notice

13 Copyright © Open Grid Forum (2009–2016). All Rights Reserved.

14 Trademarks

15 OCCI is a trademark of the Open Grid Forum.

16 Abstract

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

20 Contents

21	1 Introduction	3
22	2 Notational Conventions	3
23	3 Terms and definitions	4
24	4 OCCI Core	4
25	5 OCCI Core Model	5
26	5.1 Overview	5
27	5.2 Mutability	6
28	5.3 Classification and Identification	6
29	5.3.1 Category	6
30	5.3.2 Attribute	7
31	5.3.3 Kind	8
32	5.3.4 Mixin	9
33	5.3.5 Action	10
34	5.3.6 Instantiation	11
35	5.3.7 Templates	11
36	5.3.8 Collections	11
37	5.3.9 Discovery	12
38	5.4 The OCCI Core Base Types	12
39	5.4.1 Entity	12
40	5.4.2 Resource	13
41	5.4.3 Link	13
42	5.5 Extensibility	14
43	5.5.1 Category instances	14
44	5.5.2 Sub-typing	14
45	5.5.3 Mix-ins	14
46	6 Security Considerations	15
47	7 Glossary	15
48	8 Contributors	16
49	9 Intellectual Property Statement	16
50	10 Disclaimer	16
51	11 Full Copyright Notice	16
52	A Change Log	18
53	A.1 Action definition	18
54	A.2 Rename “resource instance” to “entity instance”	18

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 IaaS¹ 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. OCCI interaction occurs through *renderings* (including associated behaviors) and is expandable 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].

¹Infrastructure 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 an **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 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 an **entity instance**. See section 5.3.7.

type A **type** refers 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 type** or **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.

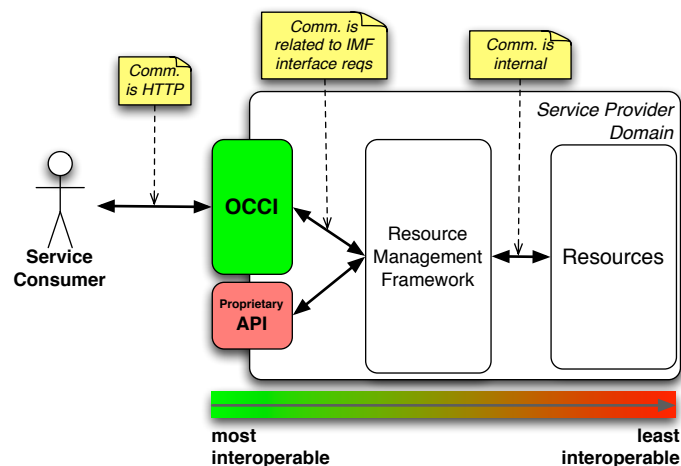


Figure 1. OCCI's place in a provider's architecture.

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. Renderings (including associated behaviors) can interact with it and extensions can expand it. 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 mixin-like concept.

Mixins first appeared in the Symbolics' object-oriented Flavors [2] 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.

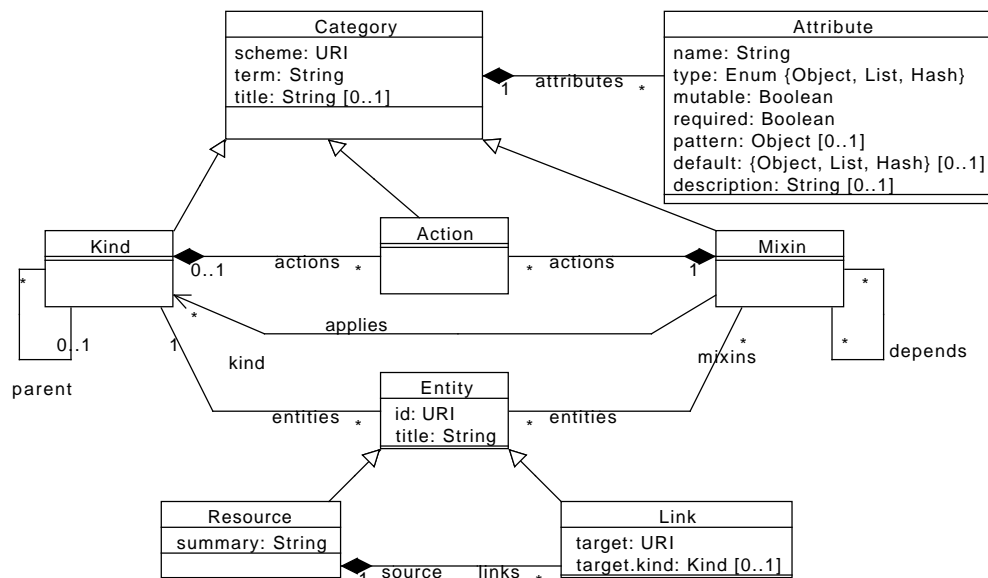


Figure 2. UML class diagram of the OCCI Core Model. The diagram provides an overview of the OCCI Core Model but is not a standalone definition thereof.

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>.

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

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

142 **Attributes** describe the name and properties of attributes found in **Entity** and its sub-types.

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

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

147 5.2 Mutability

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

154 5.3 Classification and Identification

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

159 The classification system can be summarized with the following key features:

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

173 5.3.1 Category

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

Table 2. Model attributes defined for the **Category** type.

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

178 A **Category** instance is uniquely identified by concatenating the categorization scheme with the category term,
 179 e.g., `http://example.com/category/scheme#term`. This is done to enable discovery of **Category** definitions in
 180 text-based renderings such as the OCCI Text Rendering [3]. All renderings MUST make use of and understand
 181 concatenated unique type identifiers of **Category** instances. Sub-types of **Category** such as **Kind**, **Mixin** and
 182 **Action** inherit this property.

183 The categorization schemes defined in the OCCI specification all use the `http://schemas.ogf.org/occi/` base
 184 URL. This base URL is reserved for OCCI and MUST NOT be used by service provider extensions.

185 A **Category** instance³ has zero or more associated **Attribute** instances. Each **Attribute**, see section 5.3.2,
 186 describes the name and properties of a single attribute.

187 5.3.2 Attribute

188 The **Attribute** type has a composite relationship to **Category** and defines the name and properties of client
 189 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	Value Multiplicity	Required	Client Mutability	Description
name	String	1	Yes	Immutable	Attribute name.
type	Enum {Object, List, Hash}	1	Yes	Immutable	Attribute type.
mutable	Boolean	1	Yes	Immutable	Attribute mutability.
required	Boolean	1	Yes	Immutable	Whether the Attribute must be supplied by the client at instance creation-time.
pattern	Object	0..1	–	Immutable	Attribute pattern expressed in a rendering-specific way.
default	{Object, List, Hash}	0..1	–	Immutable	Attribute default value.
description	String	0..1	–	Immutable	A description of the Attribute.

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

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

195 **type** The type of the **Attribute**. The types supported are “Object”, “List” and “Hash”.

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

197 **required** If an **Attribute** is “required” a client MUST specify a value at instance creation-time.

198 **pattern** MAY be specified in a rendering-specific format, places additional restrictions on acceptable attribute
 199 values. Detailed information is provided in every OCCI rendering document.

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

³Also applies to **Kind**, **Mixin** and **Action** instances.

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

205 5.3.3 Kind

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

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

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

215 In the initial instantiation of the OCCI Core Model, with no core model extensions, three instances of **Kind** will
 216 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	Value Multiplicity	Required	Client Mutability	Description
actions	List of Action	0..*	–	Immutable	List 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	List of Entity	0..*	–	Immutable	List of Entity instances. Instances of the particular Entity sub-type which is uniquely identified by this Kind instance.

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

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

226 **Kind Relationships** A relationship between **Kind** instances is defined by the “parent” attribute. This implies
 227 a setup of a hierarchy where the capabilities of the parent MUST be inherited by the child **Kind** instance.

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

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

⁴<http://schemas.ogf.org/occi/core#entity>

⁵The **Compute** type is defined in the OCCI Infrastructure document [4].

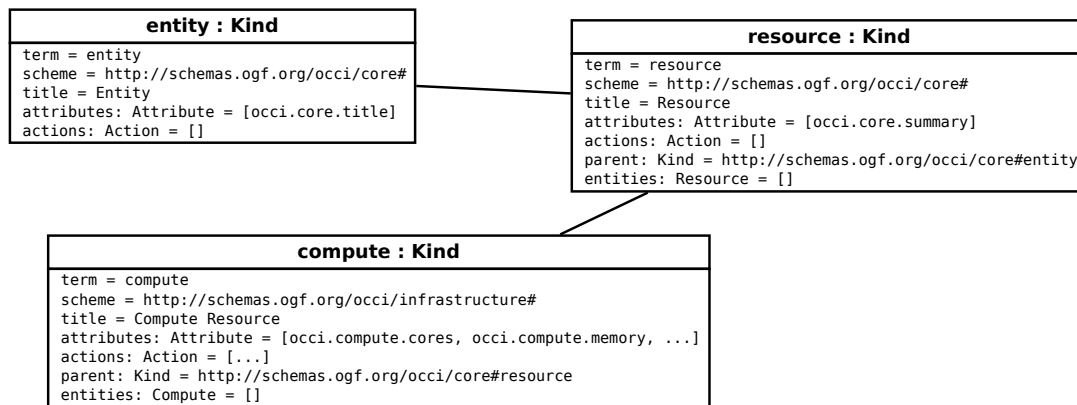


Figure 3. Object diagram illustrating the **Kind** instances involved for the **Entity**, **Resource** and **Compute** types. The **Compute** type is an extension to the OCCI Core Model defined in the OCCI Infrastructure document [4].

232 **5.3.4 Mixin**

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

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

240 A **Mixin** instance MAY be associated with an entity instance either at instance creation time or at run time.
 241 Restrictions on which entity instances a particular **Mixin** can be associated with MUST be advertised through
 242 the **Mixin.applies** model attribute.

243 When a client attempts to associate a **Mixin** instance to an entity instance at a stage not supported by a
 244 particular provider’s OCCI implementation, the provider MUST notify the client it has issued a bad request.
 245 For example a “bandwidth” **Mixin** may only be applicable to instances of the **Network**⁶ type. An OCCI provider
 246 MUST advertise such a restriction by setting **Mixin.applies** to the **Kind** instance of the **Network** type⁷.

Table 5. Model attributes defined for the **Mixin** type.

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

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

- 250 • A **Mixin** instance MUST only be associated with entity *instances*, not types, either at creation time or at
 251 run time.

⁶The **Network** type is defined in OCCI Infrastructure [4].

⁷<http://schemas.ogf.org/occi/infrastructure#network>

- 252 • A **Mixin** instance is *only* a type identifier. It MUST NOT provide the implementation of the new
253 capabilities it introduces. For example, a **Mixin** instance never contains the value of an OCCI **Attribute**.
- 254 • A **Mixin** instance MAY introduce additional **Attributes** when applied to an entity instance. The name and
255 properties of those **Attributes** MUST be exposed through **Mixin.attributes** inherited from **Category**.
256 E.g., a Location **Mixin** defining the “com.example.location” **Attribute** MUST have Location.attributes
257 populated with a single **Attribute** instance where **Attribute.name** is ‘ ‘com.example.location’ ’.
- 258 • A **Mixin** instance MAY define **Action** instances that will identify additional invocable operations on
259 any entity instance associated with the **Mixin**. **Actions** defined by a **Mixin** are exposed through the
260 **Mixin.actions** model attribute that represents the association between a **Mixin** instance and the **Action**
261 instances it defines.
- 262 • A **Mixin** instance MAY depend on another **Mixin** instance. If **Mixin B** depends on **Mixin A**, any entity
263 instance associated with **Mixin B** will receive the capabilities defined by both **Mixin B** and **Mixin A**. See
264 **Mixin Relationships** below.
- 265 • A **Mixin** instance defining no additional capabilities is considered to be a tag.
- 266 • A **Mixin** instance MAY be used as a template. A template defines default values for **Attributes** to be
267 applied at entity instance creation-time. See section 5.3.7.
- 268 • A **Mixin** instance MAY restrict which **Kind** instances it applies to using the **applies** model attribute.
269 If **Mixin.applies** is unspecified the **Mixin** may be associated to any entity instance, i.e., equivalent of
270 having **Mixin.applies** set to the **Kind** instance of **Entity**.

271 **Mixin Relationships** Other **Mixin** instances MAY depend on any given **Mixin** instance. **Mixin** relationships
272 are implemented using the **Mixin.depends** model attribute. For example a set of operating system templates,
273 implemented as **Mixin** instances, could be related to an “OS-template” **Mixin** in order to help identification.
274 **Attributes** and **Actions** defined by different **Mixin** instances are *combined* when **Mixin** relationships are present.
275 Therefore an entity instance associated with a particular **Mixin** will receive the additional capabilities defined
276 by any related **Mixin** instances as well as those defined by the **Mixin** associated.

277 5.3.5 Action

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

282 The **Action** type inherits the **Category** type. To be compliant the **Action** type MUST implement the inherited
283 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(‘ ‘size’ ’)

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

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

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

295 Table 6 shows an example of a “resize” operation defined for a Storage instance. The operation has a
296 “size” parameter which represent the size argument of the resize operation. In that example the identifying
297 **Action** instance would have **Action.attributes** populated with an **Attribute** instance where **Attribute.name** =
298 ‘‘size’’.

299 5.3.6 Instantiation

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

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

306 5.3.7 Templates

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

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

311 A **Mixin** instance MAY provide default values for **Attributes** already defined by a **Kind**. A **Mixin's At-**
312 **tribute.default** overrides the default specified by the **Kind**.

313 The handling of **Mixins** with a common (transitive) parent **Mixin**, if assigned repeatedly, MAY be defined
314 case-by-case. A new **Mixin** may, e.g., replace the previous one, be rejected, or be place alongside the previous
315 one. An example of this is the definition of replacing Resource Templates in [4].

316 5.3.8 Collections

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

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

322 A **Kind** instance maintains the collection of all entity instances of the type identified by the **Kind**.

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

327 **Adding an entity instance** to a collection is accomplished by associating the entity instance to the corre-
328 sponding **Mixin** instance.

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

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

- 333 • Retrieve the whole collection.
- 334 • Retrieve a specific item in a collection.
- 335 • Retrieve a subset of a collection.

336 The details of collection navigation is rendering specific.

337 5.3.9 Discovery

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

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

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

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

351 5.4 The OCCI Core Base Types

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

354 An instance of the **Resource** type, the **Link** type or one of their sub-types is called a *entity instance*. Each entity
355 instance within an OCCI system MUST have a unique identifier⁸ stored in the **id** model attribute of the **Entity**
356 type, as defined in table 7. The structure of these identifiers is opaque and the system should not assume
357 a static, pre-determined scheme for their structure other than the rules imposed by the Uniform Resource
358 Identifier (URI) [5] syntax.

359 Although every unique entity instance identifier MUST be a valid URI it is RECOMMENDED to use the
360 Uniform Resource Name (URN) [6] syntax.

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

362 5.4.1 Entity

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

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

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

Table 7. Model attributes defined for the **Entity** type.

Model attribute	Type	Value Multiplicity	Required	Client Mutability	Description
id	URI	1	Yes	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	Yes	Immutable	The Kind instance uniquely identifying the particular Entity sub-type of this instance.
mixins	List of Mixin	0..*	–	Mutable	Mixin instances associated to this entity instance. Consumers can expect the Attributes and Actions of the associated Mixins to be exposed by the instance.

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

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

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

372 5.4.2 Resource

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

Table 8. Model attributes defined for the **Resource** type.

Model attribute	Type	Value Multiplicity	Required	Client Mutability	Description
links	List of Link	0..*	–	Mutable	List 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 summarizing description of the Resource instance.

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

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

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

380 5.4.3 Link

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

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

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

386 The **source** attribute of a **Link** instance MUST refer to a **Resource instance** within the service provider's
 387 namespace. The **Link's target** attribute MUST point to a resource instance either within the provider's

Table 9. Model attributes defined for the **Link** type.

Model attribute	Type	Value Multiplicity	Required	Client Mutability	Description
source	Resource	1	Yes	Mutable	The Resource instances the Link instance originates from.
target	URI	1	Yes	Mutable	The unique identifier of an Object this Link instance points to.
target.kind	Kind	0..1	–	Mutable	The Kind of target, if applicable.

388 namespace or outside, hosted by a third-party. `target.kind` MAY be used to explicitly define the **Kind** of the
 389 **Resource** instance referenced by `target`. The source **Kind** is implied by the assigned **Resource** instance.

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

391 5.5 Extensibility

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

394 The OCCI Core Model can be extended using three different methods: provider-specific category instances,
 395 sub-typing and mix-ins. Custom sub-typing requires provider-specific **Kind** instances and custom mix-ins require
 396 provider-specific **Mixin** instances. Both methods MAY involve the use of provider-specific **Action** instances.
 397 The following sections define the rules for extending the OCCI Core Model.

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

399 5.5.1 Category instances

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

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

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

408 5.5.2 Sub-typing

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

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

414 5.5.3 Mix-ins

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

417 In order to support user-defined tags⁹ an OCCI implementation must allow custom **Mixin** instances to be
 418 created and destroyed by request of a client. There is no limitation in the OCCI Core Model from doing so but
 419 it is RECOMMENDED to assign a separate **Category** scheme for each user’s **Mixin** instances (e.g., per-user
 420 schemes).

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

421 6 Security Considerations

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

428 Concrete security mechanisms and protection against attacks SHOULD be specified by OCCI rendering specifi-
 429 cation. In any case, OCCI rendering specifications MUST address transport level security and authentication
 430 on the protocol level.

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

433 7 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 <i>entity instance</i> 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.
434 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 <i>entity instance</i> . The “mix-in” concept as used by OCCI <i>only</i> 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 <i>entity instance</i> . This term is considered obsolete.
tag	A Mixin instance with no attributes or actions defined. Used for taxonomic organization 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.

435

436 8 Contributors

437 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
438 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

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

440 9 Intellectual Property Statement

441 The OGF takes no position regarding the validity or scope of any intellectual property or other rights that
 442 might be claimed to pertain to the implementation or use of the technology described in this document or the
 443 extent to which any license under such rights might or might not be available; neither does it represent that
 444 it has made any effort to identify any such rights. Copies of claims of rights made available for publication
 445 and any assurances of licenses to be made available, or the result of an attempt made to obtain a general
 446 license or permission for the use of such proprietary rights by implementers or users of this specification can be
 447 obtained from the OGF Secretariat.

448 The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications,
 449 or other proprietary rights which may cover technology that may be required to practice this recommendation.
 450 Please address the information to the OGF Executive Director.

451 10 Disclaimer

452 This document and the information contained herein is provided on an "As Is" basis and the OGF disclaims all
 453 warranties, express or implied, including but not limited to any warranty that the use of the information herein
 454 will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

455 11 Full Copyright Notice

456 Copyright © Open Grid Forum (2009-2016). All Rights Reserved.

457 This document and translations of it may be copied and furnished to others, and derivative works that comment
 458 on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in
 459 whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph
 460 are included as references to the derived portions on all such copies and derivative works. The published OGF
 461 document from which such works are derived, however, may not be modified in any way, such as by removing
 462 the copyright notice or references to the OGF or other organizations, except as needed for the purpose of

463 developing new or updated OGF documents in conformance with the procedures defined in the OGF Document
464 Process, or as required to translate it into languages other than English. OGF, with the approval of its board,
465 may remove this restriction for inclusion of OGF document content for the purpose of producing standards in
466 cooperation with other international standards bodies.

467 The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or
468 assignees.

469 References

- 470 [1] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119 (Best Current Practice),
471 Internet Engineering Task Force, Mar. 1997. [Online]. Available: <http://www.ietf.org/rfc/rfc2119.txt>
- 472 [2] D. A. Moon, "Object-oriented programming with flavors," *SIGPLAN Not.*, vol. 21, pp. 1–8, June 1986.
473 [Online]. Available: <http://doi.acm.org/10.1145/960112.28698>
- 474 [3] T. Metsch and A. Edmonds, "Open Cloud Computing Interface – Text Rendering," Open Grid Forum,
475 September 2016. [Online]. Available: <https://www.ogf.org/documents/GFD.229.pdf>
- 476 [4] —, "Open Cloud Computing Interface – Infrastructure," Open Grid Forum, September 2016. [Online].
477 Available: <https://www.ogf.org/documents/GFD.224.pdf>
- 478 [5] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic
479 Syntax," RFC 3986 (Standard), Internet Engineering Task Force, Jan. 2005. [Online]. Available:
480 <http://www.ietf.org/rfc/rfc3986.txt>
- 481 [6] R. Moats, "URN Syntax," RFC 2141 (Proposed Standard), Internet Engineering Task Force, May 1997.
482 [Online]. Available: <http://www.ietf.org/rfc/rfc2141.txt>

483 A Change Log

484 The corrections introduced by the September 5, 2016 update are summarized below. The following sub-
485 sections describe the possible impact of the corrections on existing implementations and associated dependent
486 specifications such as OCCI Text Rendering [3] and OCCI Infrastructure [4].

- 487 • Adjusted language and types in Model Attribute tables (using lists where applicable).
- 488 • Added optional typed `Link` (via the `target.kind` attribute).
- 489 • Introduce an explicit `Attribute` type to expose the discoverable attribute properties already defined for
490 the OCCI base types `Entity`, `Resource`, `Link` and their sub-types.
- 491 • Correct the previously unclear definition of OCCI `Action`. The `Action` type inherits `Category` and is only
492 an identifier of an invocable operation. It does *not* represent the operation itself. The `Action` definition
493 now aligns with its use in the OCCI Text Rendering [3].
- 494 • Clarify the format of the unique entity instance identifier defined in OCCI `Entity`. Incorporate the
495 description and recommendations from the OCCI Text Rendering [3].
- 496 • Clarify that an OCCI `Mixin` instance is only a type identifier. The Core Model does not specify how a
497 mixed-in attribute is implemented. The `Mixin` instance only states that the attribute exists.
- 498 • Rename the term *resource instance* to *entity instance*. An *entity instance* refers to an instance of either
499 OCCI `Resource`, OCCI `Link` or a sub-type of either type. The *resource instance* term, while defined
500 identically, was due to its name a source of misinterpretations in the specification.
- 501 • Rename `Kind.related` to `Kind.parent` and `Mixin.related` to `Mixin.depends`. Clarify the use of `Kind`
502 and `Mixin` relationships.
- 503 • Add a new model attribute `Mixin.applies` to optionally advertise which entity instances a `Mixin` instance
504 may be associated to.

505 A.1 Action definition

506 The corrected definition of OCCI `Action` has impact neither on discovery nor on invocation of `Actions` in
507 existing implementations. The OCCI Text Rendering [3] is better aligned with OCCI Core after the corrections
508 since it already uses `type='action'` in its rendering of categories.

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

510 The change is editorial and does not affect existing implementations. The glossary contains both terms for
511 compatibility with the OCCI Text Rendering [3] specification.