

OCF BRIDGING SPECIFICATION

Version 1.3.0

Open Connectivity Foundation (OCF)
admin@openconnectivity.org

Legal Disclaimer

5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

THIS IS A DRAFT SPECIFICATION ONLY AND HAS NOT BEEN ADOPTED BY THE OPEN CONNECTIVITY FOUNDATION. THIS DRAFT SPECIFICATION MAY NOT BE RELIED UPON FOR ANY PURPOSE OTHER THAN REVIEW OF THE CURRENT STATE OF THE DEVELOPMENT OF THIS DRAFT SPECIFICATION. THE OPEN CONNECTIVITY FOUNDATION AND ITS MEMBERS RESERVE THE RIGHT WITHOUT NOTICE TO YOU TO CHANGE ANY OR ALL PORTIONS HEREOF, DELETE PORTIONS HEREOF, MAKE ADDITIONS HERETO, DISCARD THIS DRAFT SPECIFICATION IN ITS ENTIRETY OR OTHERWISE MODIFY THIS DRAFT SPECIFICATION AT ANY TIME. YOU SHOULD NOT AND MAY NOT RELY UPON THIS DRAFT SPECIFICATION IN ANY WAY, INCLUDING BUT NOT LIMITED TO THE DEVELOPMENT OF ANY PRODUCTS OR SERVICES. IMPLEMENTATION OF THIS DRAFT SPECIFICATION IS DONE AT YOUR OWN RISK AMEND AND IT IS NOT SUBJECT TO ANY LICENSING GRANTS OR COMMITMENTS UNDER THE OPEN CONNECTIVITY FOUNDATION INTELLECTUAL PROPERTY RIGHTS POLICY OR OTHERWISE. IN CONSIDERATION OF THE OPEN CONNECTIVITY FOUNDATION GRANTING YOU ACCESS TO THIS DRAFT SPECIFICATION, YOU DO HEREBY WAIVE ANY AND ALL CLAIMS ASSOCIATED HEREWITH INCLUDING BUT NOT LIMITED TO THOSE CLAIMS DISCUSSED BELOW, AS WELL AS CLAIMS OF DETRIMENTAL RELIANCE.

The OCF logo is a trademark of Open Connectivity Foundation, Inc. in the United States or other countries. *Other names and brands may be claimed as the property of others.

Copyright © 2017 Open Connectivity Foundation, Inc. All rights reserved.

Copying or other form of reproduction and/or distribution of these works are strictly prohibited.

29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

CONTENTS

1	Scope	6
2	Normative references	6
3	Terms, definitions, symbols and abbreviations	7
3.1	Terms and definitions	7
3.2	Symbols and abbreviations	9
3.3	Conventions	9
4	Document conventions and organization	9
4.1	Notation.....	9
4.2	Data types	10
4.3	Document structure	10
5	Operational Scenarios.....	10
5.1	“Deep translation” vs. “on-the-fly”	10
5.2	Use of introspection.....	11
5.3	Stability and loss of data	11
6	OCF Bridge Device	12
6.1	Resource Discovery.....	12
6.2	General Requirements.....	22
6.3	Security.....	22
6.3.1	Blocking communication of Bridged Devices with the OCF ecosystem	23
7	AllJoyn Translation.....	23
7.1	Requirements Specific to an AllJoyn Translator	23
7.1.1	Exposing AllJoyn producer devices to OCF Clients	23
7.1.2	Exposing OCF resources to AllJoyn consumer applications	31
7.2	On-the-Fly Translation from D-Bus and OCF payloads.....	37
7.2.1	Translation without aid of introspection	37
7.2.2	Translation with aid of introspection	43
8	Device Type Definitions.....	48
9	Resource Type definitions	48
9.1	List of resource types	48
9.2	Secure Mode	48
9.2.1	Introduction	48
9.2.2	Example URI Path.....	48
9.2.3	Resource Type	49
9.2.4	RAML Definition	49
9.2.5	Swagger2.0 Definition	51
9.2.6	Property Definition	53
9.2.7	CRUDN behaviour.....	53
9.3	AllJoyn Object	53
9.3.1	Introduction	53

71	9.3.2	Example URI Path.....	53
72	9.3.3	Resource Type.....	53
73	9.3.4	RAML Definition.....	53
74	9.3.5	Swagger2.0 Definition.....	55
75	9.3.6	CRUDN behaviour.....	58
76			

DRAFT

77
78
79
80

Figures

Figure 1. OCF Bridge Device Components	7
Figure 2: Schematic overview of an OCF Bridge Device bridging non-OCF devices	12

DRAFT

Tables

81		
82	Table 1: oic.wk.d resource type definition	26
83	Table 2: oic.wk.con resource type definition	28
84	Table 3: oic.wk.p Resource Type definition.....	29
85	Table 4: oic.wk.con.p Resource Type definition	31
86	Table 5: AllJoyn About Data fields	33
87	Table 6: AllJoyn Configuration Data fields	36
88	Table 7 Alphabetical list of resource types.....	48
89		
90		

DRAFT

91 **1 Scope**

92 This document specifies a framework for translation between OCF devices and other ecosystems,
93 and specifies the behaviour of a translator that exposes AllJoyn producer applications to OCF
94 clients, and exposes OCF servers to AllJoyn consumer applications. Translation of specific AllJoyn
95 interfaces to or from specific OCF resource types is left to other specifications. Translation of
96 protocols other than AllJoyn is left to a future version of this specification. This document provides
97 generic requirements that apply unless overridden by a more specific document.

98 **2 Normative references**

99 The following documents, in whole or in part, are normatively referenced in this document and are
100 indispensable for its application. For dated references, only the edition cited applies. For undated
101 references, the latest edition of the referenced document (including any amendments) applies.

102 AllJoyn About Interface Specification, *About Feature Interface Definitions*, Version 14.12
103 <https://allseenalliance.org/framework/documentation/learn/core/about-announcement/interface>

104 AllJoyn Configuration Interface Specification, *Configuration Interface Definition*, Version 14.12
105 <https://allseenalliance.org/framework/documentation/learn/core/configuration/interface>

106 D-Bus Specification, *D-Bus Specification*
107 <https://dbus.freedesktop.org/doc/dbus-specification.html>

108 IEEE 754, *IEEE Standard for Floating-Point Arithmetic*, August 2008

109 IETF RFC 4122, *A Universally Unique IDentifier (UUID) URN Namespace*, July 2005
110 <https://www.rfc-editor.org/info/rfc4122>

111 IETF RFC 4648, *The Base16, Base32, and Base64 Data Encodings*, October 2006
112 <https://www.rfc-editor.org/info/rfc4648>

113 IETF RFC 6973, *Privacy Considerations for Internet Protocols*, July 2013
114 <https://www.rfc-editor.org/info/rfc6973>

115 IETF RFC 7049, *Concise Binary Object Representation (CBOR)*, October 2013
116 <https://www.rfc-editor.org/info/rfc7049>

117 IETF RFC 7159, *The JavaScript Object Notation (JSON) Data Interchange Format*, March 2014
118 <https://www.rfc-editor.org/info/rfc7159>

119 JSON Schema Core, *JSON Schema: core definitions and terminology*, January 2013
120 <http://json-schema.org/latest/json-schema-core.html>

121 JSON Schema Validation, *JSON Schema: interactive and non-interactive validation*, January
122 2013
123 <http://json-schema.org/latest/json-schema-validation.html>

124 [JSON Hyper-Schema, *JSON Hyper-Schema: A Vocabulary for Hypermedia Annotation of JSON*,
125 October 2016
126 <http://json-schema.org/latest/json-schema-hypermedia.html>](http://json-schema.org/latest/json-schema-hypermedia.html)

127 OCF Core Specification, *Open Connectivity Foundation Core Specification*, Version 1.0

128 OCF Security Specification, *Open Connectivity Foundation Security Specification*, Version 1.0

129 OCF Resource to AllJoyn Interface Mapping Specification, *Open Connectivity Foundation
130 Resource to AllJoyn Interface Mapping Specification*, Version 1.0

131 OIC 1.1 Core Specification, *Open Interconnect Consortium Core Specification*, Version 1.1

132 RAML Specification, *RESTful API Modeling Language*, Version 0.8

133 <https://github.com/raml-org/raml-spec/blob/master/versions/raml-08/raml-08.md>

134 OpenAPI specification, Version 2.0

135 <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md>

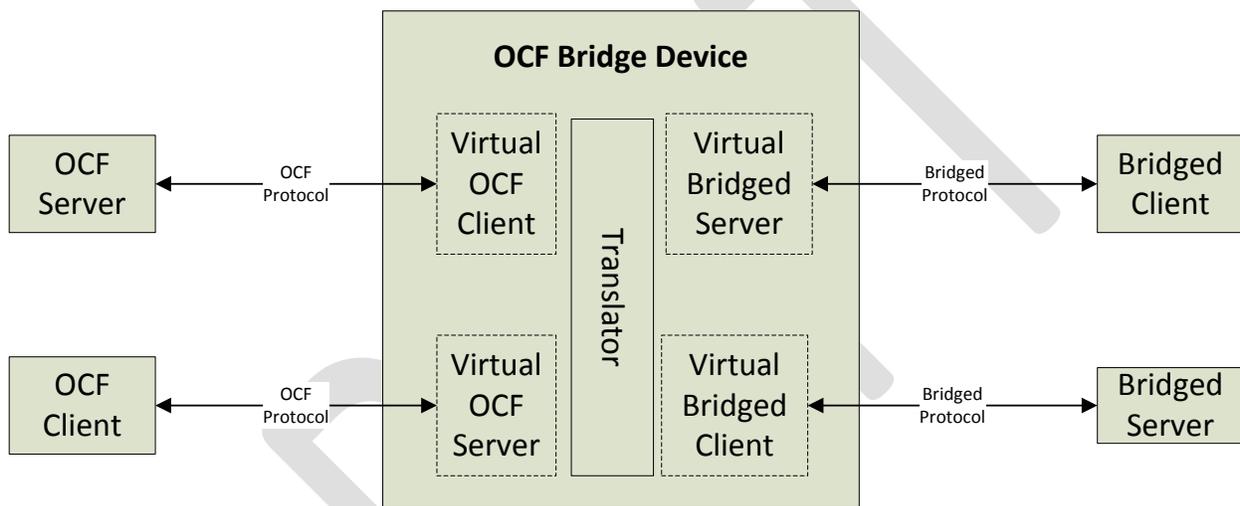
136 3 Terms, definitions, symbols and abbreviations

137 3.1 Terms and definitions

138 3.1.1

139 OCF Bridge Device

140 An OCF Device that can represent devices that exist on the network but communicate using a
141 Bridged Protocol rather than OCF protocols.



142

143

144

Figure 1. OCF Bridge Device Components

145 3.1.2

146 Bridged Protocol

147 another protocol (e.g., AllJoyn) that is being translated to or from OCF protocols

148 3.1.3

149 Translator

150 an OCF Bridge Device component that is responsible for translating to or from a specific Bridged
151 Protocol. More than one translator can exist on the same OCF Bridge Device, for different Bridged
152 Protocols.

153 3.1.4

154 OCF Client

155 a logical entity that accesses an OCF Resource on an OCF Server, which might be a Virtual OCF
156 Server exposed by the OCF Bridge Device.

157 3.1.5

158 Bridged Client

159 a logical entity that accesses data via a Bridged Protocol. For example, an AllJoyn Consumer
160 application is a Bridged Client.

161 **3.1.6**
162 **Virtual OCF Client**
163 a logical representation of a Bridged Client, which an OCF Bridge Device exposes to OCF Servers.

164 **3.1.7**
165 **Virtual Bridged Client**
166 a logical representation of an OCF Client, which an OCF Bridge Device exposes to Bridged Servers.

167 **3.1.8**
168 **OCF Device**
169 a logical entity that assumes one or more OCF roles (OCF Client, OCF Server). More than one
170 OCF Device can exist on the same physical platform.

171 **3.1.9**
172 **Virtual OCF Server**
173 a logical representation of a Bridged Server, which an OCF Bridge Device exposes to OCF Clients.

174 **3.1.10**
175 **Bridged Server**
176 a logical entity that provides data via a Bridged Protocol. For example, an AllJoyn Producer is a
177 Bridged Server. More than one Bridged Server can exist on the same physical platform.

178 **3.1.11**
179 **Virtual Bridged Server**
180 a logical representation of an OCF Server, which an OCF Bridge Device exposes to Bridged Clients.

181 **3.1.12**
182 **OCF Resource**
183 represents an artifact modelled and exposed by the OCF Framework

184 **3.1.13**
185 **Virtual OCF Resource**
186 a logical representation of a Bridged Resource, which an OCF Bridge Device exposes to OCF
187 Clients.

188 **3.1.14**
189 **Bridged Resource**
190 represents an artifact modelled and exposed by a Bridged Protocol. For example, an AllJoyn
191 object is a Bridged Resource.

192 **3.1.15**
193 **OCF Resource Property**
194 a significant aspect or notion including metadata that is exposed through the OCF Resource

195 **3.1.16**
196 **OCF Resource Type**
197 an OCF Resource Property that represents the data type definition for the OCF Resource

198 **3.1.17**
199 **Bridged Resource Type**
200 a schema used with a Bridged Protocol. For example, AllJoyn Interfaces are Bridged Resource
201 Types.

202 **3.1.18**
203 **OCF Server**
204 a logical entity with the role of providing resource state information and allowing remote control of
205 its resources.

206 **3.1.19**
207 **Onboarding Tool**
208 defined by the OCF Security Specification as: A logical entity within a specific IoT network that
209 establishes ownership for a specific device and helps bring the device into operational state within
210 that network.

211 **3.1.20**
212 **Bridged Device**
213 a Bridged Client or Bridged Server.

214 **3.1.21**
215 **Virtual OCF Device**
216 a Virtual OCF Client or Virtual OCF Server.

217 **3.2 Symbols and abbreviations**

218 **3.2.1**
219 **CRUDN**
220 Create Read Update Delete Notify
221 indicating which operations are possible on the resource

222 **3.2.2**
223 **CSV**
224 Comma Separated Value List
225 construction to have more fields in 1 string separated by commas. If a value contains a comma,
226 then the comma can be escaped by adding “\” in front of the comma.

227 **3.2.3**
228 **OCF**
229 Open Connectivity Foundation
230 organization that created these specifications

231 **3.2.4**
232 **RAML**
233 RESTful API Modeling Language
234 Simple and succinct way of describing practically RESTful APIs (see the RAML Specification)

235 **3.3 Conventions**

236 In this specification several terms, conditions, mechanisms, sequences, parameters, events,
237 states, or similar terms are printed with the first letter of each word in uppercase and the rest
238 lowercase (e.g., Network Architecture). Any lowercase uses of these words have the normal
239 technical English meaning.

240 **4 Document conventions and organization**

241 For the purposes of this document, the terms and definitions given in the OCF 1.0 Core
242 Specification apply.

243 **4.1 Notation**

244 In this document, features are described as required, recommended, allowed or DEPRECATED as
245 follows:

246 Required (or shall or mandatory).

247 – These basic features shall be implemented to comply with this specification. The phrases “shall
248 not”, and “PROHIBITED” indicate behaviour that is prohibited, i.e. that if performed means the
249 implementation is not in compliance.

250 Recommended (or should).

- 251 – These features add functionality supported by this specification and should be implemented.
252 Recommended features take advantage of the capabilities of this specification, usually without
253 imposing major increase of complexity. Notice that for compliance testing, if a recommended
254 feature is implemented, it shall meet the specified requirements to be in compliance with these
255 guidelines. Some recommended features could become requirements in the future. The phrase
256 “should not” indicates behaviour that is permitted but not recommended.

257 Allowed (or allowed).

- 258 – These features are neither required nor recommended, but if the feature is implemented, it
259 shall meet the specified requirements to be in compliance with these guidelines.

260 Conditionally allowed (CA)

- 261 – The definition or behaviour depends on a condition. If the specified condition is met, then the
262 definition or behaviour is allowed, otherwise it is not allowed.

263 Conditionally required (CR)

- 264 – The definition or behaviour depends on a condition. If the specified condition is met, then the
265 definition or behaviour is required. Otherwise the definition or behaviour is allowed as default
266 unless specifically defined as not allowed.

267 DEPRECATED

- 268 – Although these features are still described in this specification, they should not be implemented
269 except for backward compatibility. The occurrence of a deprecated feature during operation of
270 an implementation compliant with the current specification has no effect on the
271 implementation’s operation and does not produce any error conditions. Backward compatibility
272 may require that a feature is implemented and functions as specified but it shall never be used
273 by implementations compliant with this specification.

274 Strings that are to be taken literally are enclosed in “double quotes”.

275 Words that are emphasized are printed in *italic*.

276 **4.2 Data types**

277 Data types are defined in the OCF 1.0 Core Specification.

278 **4.3 Document structure**

279 Section 5 discusses operational scenarios. Section 6 covers generic requirements for any OCF
280 Bridge, and section 7 covers the specific requirements for a Bridge that translates to/from AllJoyn.
281 These are covered separately to ease the task of defining translation to other protocols in the
282 future.

283 **5 Operational Scenarios**

284 The overall goals are to:

- 285 1. make Bridged Servers appear to OCF clients as if they were native OCF servers, and
- 286 2. make OCF servers appear to Bridged Clients as if they were native non-OCF servers.

287 **5.1 “Deep translation” vs. “on-the-fly”**

288 When translating a service between a Bridged Protocol (e.g., AllJoyn) and OCF protocols, there
289 are two possible types of translation. Translators are expected to dedicate most of their logic to
290 “deep translation” types of communication, in which data models used with the Bridged Protocol

291 are mapped to the equivalent OCF Resource Types and vice-versa, in such a way that a compliant
292 OCF Client or Bridged Client would be able to interact with the service without realising that a
293 translation was made.

294 “Deep translation” is out of the scope of this document, as the procedure far exceeds mapping of
295 types. For example, clients on one side of a translator may decide to represent an intensity as an
296 8-bit value between 0 and 255, whereas the devices on the other may have chosen to represent
297 that as a floating-point number between 0.0 and 1.0. It’s also possible that the procedure may
298 require storing state in the translator. Either way, the programming of such translation will require
299 dedicated effort and study of the mechanisms on both sides.

300 The other type of translation, the “on-the-fly” or “one-to-one” translation, requires no prior
301 knowledge of the device-specific schema in question on the part of the translator. The burden is,
302 instead, on one of the other participants in the communication, usually the client application. That
303 stems from the fact that “on-the-fly” translation always produces Bridged Resource Types and OCF
304 Resource Types as *vendor extensions*.

305 For AllJoyn, deep translation is specified in OCF ASA Mapping, and on-the-fly translation is
306 covered in section 7.2 of this document.

307 **5.2 Use of introspection**

308 Whenever possible, the translation code should make use of metadata available that indicates
309 what the sender and recipient of the message in question are expecting. For example, devices that
310 are AllJoyn Certified are required to carry the introspection data for each object and interface they
311 expose. The OIC 1.1 Core Specification makes no such requirement, but the OCF 1.0 Core
312 Specification does. When the metadata is available, translators should convert the incoming
313 payload to exactly the format expected by the recipient and should use information when
314 translating replies to form a more useful message.

315 For example, for an AllJoyn translator, the expected interaction list is presented on the list below:

Message Type	Sender	Receiver	Metadata
Request	AllJoyn 16.10	OIC 1.1	Not available
Request	AllJoyn 16.10	OCF 1.0	Available
Request	OIC 1.1 or OCF 1.0	AllJoyn 16.10	Available
Response	AllJoyn 16.10	OIC 1.1 or OCF 1.0	Available
Response	OIC 1.1	AllJoyn 16.10	Not available
Response	OCF 1.0	AllJoyn 16.10	Available

316 **5.3 Stability and loss of data**

317 Round-tripping through the translation process specified in this document is not expected to
318 reproduce the same original message. The process is, however, designed not to lose data or
319 precision in messages, though it should be noted that both OCF and AllJoyn payload formats allow
320 for future extensions not considered in this document.

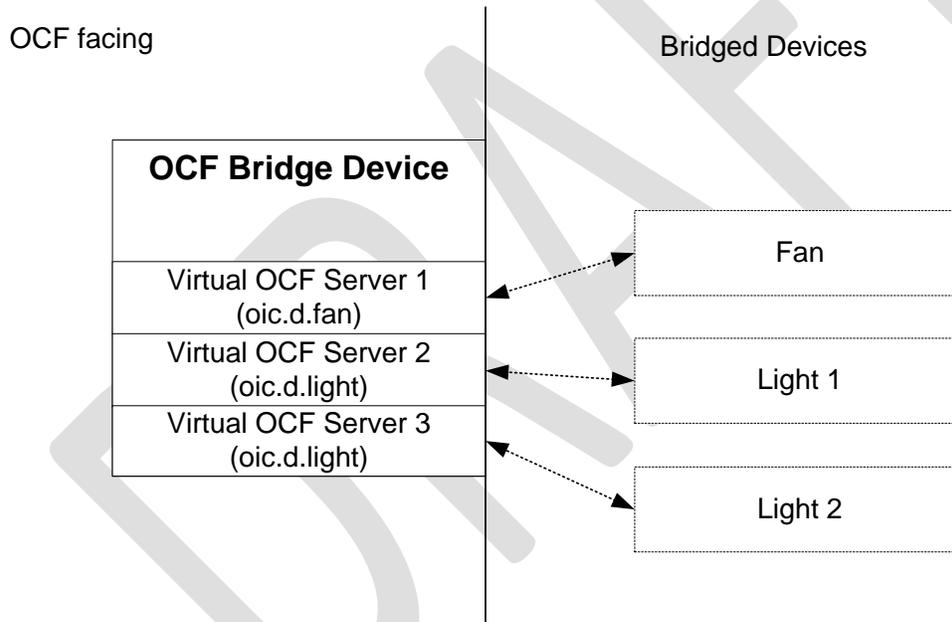
321 However, a third round of translation should produce the same identical message as was
322 previously produced, provided the same information is available. That is, in the above chain,
323 payloads 2 and 4 as well as 3 and 5 should be identical.

324 **6 OCF Bridge Device**

325 This section describes the functionality of an OCF Bridge Device; such a device is illustrated in
326 Figure 2.

327 An OCF Bridge Device is a device that represents one or more Bridged Devices as Virtual OCF
328 Devices on the network and/or represents one or more OCF Devices as Virtual Devices using
329 another protocol on the network. The Bridged Devices themselves are out of the scope of this
330 document. The only difference between a native OCF Device and a Virtual Bridged Device is how
331 the device is encapsulated in an OCF Bridge Device.

332 An OCF Bridge Device shall be indicated on the OCF network with a Device Type of "oic.d.bridge".
333 This provides to an OCF Client an explicit indication that the discovered Device is performing a
334 bridging function. This is useful for several reasons; 1) when establishing a home network the
335 Client can determine that the bridge is reachable and functional when no bridged devices are
336 present, 2) allows for specific actions to be performed on the bridge considering the known
337 functionality a bridge supports, 3) allows for explicit discovery of all devices that are serving a
338 bridging function which benefits trouble shooting and maintenance actions on behalf of a user.
339 When such a device is discovered the exposed Resources on the OCF Bridge Device describe
340 other devices. For example, as shown in Figure 2.



341
342 **Figure 2: Schematic overview of an OCF Bridge Device bridging non-OCF devices**

343 It is expected that the OCF Bridge Device creates a set of devices during the start-up of the OCF
344 Bridge Device. The exposed set of Virtual OCF Devices can change as Bridged Devices are added
345 or removed from the bridge. The adding and removing of Bridged Devices is implementation
346 dependent. When an OCF Bridge Device changes the set of exposed Virtual OCF Devices, it shall
347 notify any OCF Clients subscribed to its "/oic/res".

348 **6.1 Resource Discovery**

349 An OCF Bridge Device shall detect devices that arrive and leave the Bridged network or the OCF
350 network. Where there is no pre-existing mechanism to reliably detect the arrival and departure of
351 devices on a network, an OCF Bridge Device shall periodically poll the network to detect arrival
352 and departure of devices, for example using COAP multicast discovery (a multicast RETRIEVE of

353 "/oic/res") in the case of the OCF network. OCF Bridge Device implementations are encouraged to
354 use a poll interval of 30 seconds plus or minus a random delay of a few seconds.

355 An OCF Bridge Device shall respond to network discovery commands on behalf of the exposed
356 bridged devices. All bridged devices with all their Resources shall be listed in "/oic/res" of the
357 Bridge. The response to a RETRIEVE on "/oic/res" shall only include the devices that match the
358 RETRIEVE request.

359 The resource reference determined from each Link exposed by "/oic/res" on the Bridge shall be
360 unique. The Bridge shall meet the requirements defined in the OCF 1.0 Core Specification for
361 population of the Properties and Link parameters in "/oic/res".

362 For example, if an OCF Bridge Device exposes Virtual OCF Servers for the fan and lights shown
363 in Figure 2, the bridge might return the following information corresponding to the JSON below to
364 a legacy OIC 1.1 client doing a RETRIEVE on "/oic/res". (Note that what is returned is not in the
365 JSON format but in a suitable encoding as defined in the OCF 1.0 Core Specification.)

```
366 [
367   {
368     "di": "e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
369     "links": [
370       {
371         "href": "coap://[2001:db8:a::b1d4]:55555/oic/res",
372         "rel": "self",
373         "rt": ["oic.wk.res"],
374         "if": ["oic.if.ll", "oic.if.baseline"],
375         "p": {"bm": 3, "sec": true, "port": 11111}
376       },
377       {
378         "href": "/oic/d",
379         "rt": ["oic.wk.d", "oic.d.bridge"],
380         "if": ["oic.if.r", "oic.if.baseline"],
381         "p": {"bm": 3, "sec": true, "port": 11111}
382       },
383       {
384         "href": "/oic/p",
385         "rt": ["oic.wk.p"],
386         "if": ["oic.if.r", "oic.if.baseline"],
387         "p": {"bm": 3, "sec": true, "port": 11111}
388       },
389       {
390         "href": "/mySecureMode",
391         "rt": ["oic.r.securemode"],
392         "if": ["oic.if.rw", "oic.if.baseline"],
393         "p": {"bm": 3, "sec": true, "port": 11111}
394       },
395       {
396         "href": "/oic/sec/doxm",
397         "rt": ["oic.r.doxm"],
398         "if": ["oic.if.baseline"],
399         "p": {"bm": 1, "sec": true, "port": 11111}
400       },
401       {
402         "href": "/oic/sec/pstat",
403         "rt": ["oic.r.pstat"],
404         "if": ["oic.if.baseline"],
405         "p": {"bm": 1, "sec": true, "port": 11111}
406       },
407       {
408         "href": "/oic/sec/cred",
409
```

```

410     "rt": ["oic.r.cred"],
411     "if": ["oic.if.baseline"],
412     "p": {"bm": 1, "sec": true, "port": 11111}
413   },
414   {
415     "href": "/oic/sec/acl2",
416     "rt": ["oic.r.acl2"],
417     "if": ["oic.if.baseline"],
418     "p": {"bm": 1, "sec": true, "port": 11111}
419   },
420   {
421     "href": "/myIntrospection",
422     "rt": ["oic.wk.introspection"],
423     "if": ["oic.if.r", "oic.if.baseline"],
424     "p": {"bm": 3, "sec": true, "port": 11111}
425   }
426 ]
427 },
428 {
429   "di": "88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
430   "links": [
431     {
432       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/res",
433       "rt": ["oic.wk.res"],
434       "if": ["oic.if.ll", "oic.if.baseline"],
435       "p": {"bm": 3, "sec": true, "port": 22222}
436     },
437     {
438       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/d",
439       "rt": ["oic.wk.d", "oic.d.fan", "oic.d.virtual"],
440       "if": ["oic.if.r", "oic.if.baseline"],
441       "p": {"bm": 3, "sec": true, "port": 22222}
442     },
443     {
444       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/p",
445       "rt": ["oic.wk.p"],
446       "if": ["oic.if.r", "oic.if.baseline"],
447       "p": {"bm": 3, "sec": true, "port": 22222}
448     },
449     {
450       "href": "coaps://[2001:db8:a::b1d4]:22222/myFan",
451       "rt": ["oic.r.switch.binary"],
452       "if": ["oic.if.a", "oic.if.baseline"],
453       "p": {"bm": 3, "sec": true, "port": 22222}
454     },
455     {
456       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/sec/doxm",
457       "rt": ["oic.r.doxm"],
458       "if": ["oic.if.baseline"],
459       "p": {"bm": 1, "sec": true, "port": 22222}
460     },
461     {
462       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/sec/pstat",
463       "rt": ["oic.r.pstat"],
464       "if": ["oic.if.baseline"],
465       "p": {"bm": 1, "sec": true, "port": 22222}
466     },
467     {
468       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/sec/cred",
469       "rt": ["oic.r.cred"],
470       "if": ["oic.if.baseline"],
471       "p": {"bm": 1, "sec": true, "port": 22222}
472     }
473   ],

```

```

473     {
474         "href": "coaps://[2001:db8:a::b1d4]:22222/oic/sec/acl2",
475         "rt": ["oic.r.acl2"],
476         "if": ["oic.if.baseline"],
477         "p": {"bm": 1, "sec": true, "port": 22222}
478     },
479     {
480         "href": "coaps://[2001:db8:a::b1d4]:22222/myFanIntrospection",
481         "rt": ["oic.wk.introspection"],
482         "if": ["oic.if.r", "oic.if.baseline"],
483         "p": {"bm": 3, "sec": true, "port": 22222}
484     }
485 ]
486 },
487 {
488     "di": "dc70373c-1e8d-4fb3-962e-017eaa863989",
489     "links": [
490         {
491             "href": "coaps://[2001:db8:a::b1d4]:33333/oic/res",
492             "rt": ["oic.wk.res"],
493             "if": ["oic.if.ll", "oic.if.baseline"],
494             "p": {"bm": 3, "sec": true, "port": 33333}
495         },
496         {
497             "href": "coaps://[2001:db8:a::b1d4]:33333/oic/d",
498             "rt": ["oic.wk.d", "oic.d.light", "oic.d.virtual"],
499             "if": ["oic.if.r", "oic.if.baseline"],
500             "p": {"bm": 3, "sec": true, "port": 33333}
501         },
502         {
503             "href": "coaps://[2001:db8:a::b1d4]:33333/oic/p",
504             "rt": ["oic.wk.p"],
505             "if": ["oic.if.r", "oic.if.baseline"],
506             "p": {"bm": 3, "sec": true, "port": 33333}
507         },
508         {
509             "href": "coaps://[2001:db8:a::b1d4]:33333/myLight",
510             "rt": ["oic.r.switch.binary"],
511             "if": ["oic.if.a", "oic.if.baseline"],
512             "p": {"bm": 3, "sec": true, "port": 33333}
513         },
514         {
515             "href": "coaps://[2001:db8:a::b1d4]:33333/oic/sec/doxm",
516             "rt": ["oic.r.doxm"],
517             "if": ["oic.if.baseline"],
518             "p": {"bm": 1, "sec": true, "port": 33333}
519         },
520         {
521             "href": "coaps://[2001:db8:a::b1d4]:33333/oic/sec/pstat",
522             "rt": ["oic.r.pstat"],
523             "if": ["oic.if.baseline"],
524             "p": {"bm": 1, "sec": true, "port": 33333}
525         },
526         {
527             "href": "coaps://[2001:db8:a::b1d4]:33333/oic/sec/cred",
528             "rt": ["oic.r.cred"],
529             "if": ["oic.if.baseline"],
530             "p": {"bm": 1, "sec": true, "port": 33333}
531         },
532         {
533             "href": "coaps://[2001:db8:a::b1d4]:33333/oic/sec/acl2",
534             "rt": ["oic.r.acl2"],
535             "if": ["oic.if.baseline"],

```

```

536     "p": {"bm": 1, "sec": true, "port": 33333}
537   },
538   {
539     "href": "coaps://[2001:db8:a:b1d4]:33333/myLightIntrospection",
540     "rt": ["oic.wk.introspection"],
541     "if": ["oic.if.r", "oic.if.baseline"],
542     "p": {"bm": 3, "sec": true, "port": 33333}
543   }
544 ]
545 },
546 {
547   "di": "8202138e-aa22-452c-b512-9ebad02bef7c",
548   "links": [
549     {
550       "href": "coaps://[2001:db8:a:b1d4]:44444/oic/res",
551       "rt": ["oic.wk.res"],
552       "if": ["oic.if.ll", "oic.if.baseline"],
553       "p": {"bm": 3, "sec": true, "port": 44444}
554     },
555     {
556       "href": "coaps://[2001:db8:a:b1d4]:44444/oic/d",
557       "rt": ["oic.wk.d", "oic.d.light", "oic.d.virtual"],
558       "if": ["oic.if.r", "oic.if.baseline"],
559       "p": {"bm": 3, "sec": true, "port": 44444}
560     },
561     {
562       "href": "coaps://[2001:db8:a:b1d4]:44444/oic/p",
563       "rt": ["oic.wk.p"],
564       "if": ["oic.if.r", "oic.if.baseline"],
565       "p": {"bm": 3, "sec": true, "port": 44444}
566     },
567     {
568       "href": "coaps://[2001:db8:a:b1d4]:44444/myLight",
569       "rt": ["oic.r.switch.binary"],
570       "if": ["oic.if.a", "oic.if.baseline"],
571       "p": {"bm": 3, "sec": true, "port": 44444}
572     },
573     {
574       "href": "coaps://[2001:db8:a:b1d4]:44444/oic/sec/doxm",
575       "rt": ["oic.r.doxm"],
576       "if": ["oic.if.baseline"],
577       "p": {"bm": 1, "sec": true, "port": 44444}
578     },
579     {
580       "href": "coaps://[2001:db8:a:b1d4]:44444/oic/sec/pstat",
581       "rt": ["oic.r.pstat"],
582       "if": ["oic.if.baseline"],
583       "p": {"bm": 1, "sec": true, "port": 44444}
584     },
585     {
586       "href": "coaps://[2001:db8:a:b1d4]:44444/oic/sec/cred",
587       "rt": ["oic.r.cred"],
588       "if": ["oic.if.baseline"],
589       "p": {"bm": 1, "sec": true, "port": 44444}
590     },
591     {
592       "href": "coaps://[2001:db8:a:b1d4]:44444/oic/sec/acl2",
593       "rt": ["oic.r.acl2"],
594       "if": ["oic.if.baseline"],
595       "p": {"bm": 1, "sec": true, "port": 44444}
596     },
597     {
598       "href": "coaps://[2001:db8:a:b1d4]:44444/myLightIntrospection",

```

```

599     "rt": ["oic.wk.introspection"],
600     "if": ["oic.if.r", "oic.if.baseline"],
601     "p": {"bm": 3, "sec": true, "port": 44444}
602   }
603 ]
604 }
605 ]

```

606 The above example illustrates that each Virtual OCF Server has its own “di” and endpoint
607 exposed by the bridge, and that “/oic/p” and “/oic/d” are available for each Virtual OCF Server.

608
609 When an OCF Client requests a content format of “application/vnd.ocf+cbor”, the same bridge
610 will return information corresponding to the JSON below. (Note that what is returned is not in the
611 JSON format but in a suitable encoding as defined in the OCF 1.0 Core Specification.)

```

612 [
613 {
614   {
615     "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
616     "href": "/oic/res",
617     "rel": "self",
618     "rt": ["oic.wk.res"],
619     "if": ["oic.if.ll", "oic.if.baseline"],
620     "p": {"bm": 3},
621     "eps": [{"ep": "coap://[2001:db8:a::b1d4]:55555"},
622             {"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
623   },
624   {
625     "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
626     "href": "/oic/d",
627     "rt": ["oic.wk.d", "oic.d.bridge"],
628     "if": ["oic.if.r", "oic.if.baseline"],
629     "p": {"bm": 3},
630     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
631   },
632   {
633     "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
634     "href": "/oic/p",
635     "rt": ["oic.wk.p"],
636     "if": ["oic.if.r", "oic.if.baseline"],
637     "p": {"bm": 3},
638     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
639   },
640   {
641     "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
642     "href": "/mySecureMode",
643     "rt": ["oic.r.securemode"],
644     "if": ["oic.if.rw", "oic.if.baseline"],
645     "p": {"bm": 3},
646     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
647   },
648   {
649     "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
650     "href": "/oic/sec/doxm",
651     "rt": ["oic.r.doxm"],
652     "if": ["oic.if.baseline"],
653     "p": {"bm": 1},
654     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
655   },
656   {
657     "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
658     "href": "/oic/sec/pstat",
659     "rt": ["oic.r.pstat"],

```

```

660     "if": ["oic.if.baseline"],
661     "p": {"bm": 1},
662     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
663   },
664   {
665     "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
666     "href": "/oic/sec/cred",
667     "rt": ["oic.r.cred"],
668     "if": ["oic.if.baseline"],
669     "p": {"bm": 1},
670     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
671   },
672   {
673     "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
674     "href": "/oic/sec/acl2",
675     "rt": ["oic.r.acl2"],
676     "if": ["oic.if.baseline"],
677     "p": {"bm": 1},
678     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
679   },
680   {
681     "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
682     "href": "/myIntrospection",
683     "rt": ["oic.wk.introspection"],
684     "if": ["oic.if.r", "oic.if.baseline"],
685     "p": {"bm": 3},
686     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
687   },
688
689
690   {
691     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
692     "href": "/oic/res",
693     "rt": ["oic.wk.res"],
694     "if": ["oic.if.ll", "oic.if.baseline"],
695     "p": {"bm": 3},
696     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
697   },
698   {
699     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
700     "href": "/oic/d",
701     "rt": ["oic.wk.d", "oic.d.fan", "oic.d.virtual"],
702     "if": ["oic.if.r", "oic.if.baseline"],
703     "p": {"bm": 3},
704     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
705   },
706   {
707     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
708     "href": "/oic/p",
709     "rt": ["oic.wk.p"],
710     "if": ["oic.if.r", "oic.if.baseline"],
711     "p": {"bm": 3},
712     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
713   },
714   {
715     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
716     "href": "/myFan",
717     "rt": ["oic.r.switch.binary"],
718     "if": ["oic.if.a", "oic.if.baseline"],
719     "p": {"bm": 3},
720     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
721   },
722   {

```

```

723     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
724     "href": "/oic/sec/doxm",
725     "rt": ["oic.r.doxm"],
726     "if": ["oic.if.baseline"],
727     "p": {"bm": 1},
728     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
729   },
730   {
731     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
732     "href": "/oic/sec/pstat",
733     "rt": ["oic.r.pstat"],
734     "if": ["oic.if.baseline"],
735     "p": {"bm": 1},
736     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
737   },
738   {
739     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
740     "href": "/oic/sec/cred",
741     "rt": ["oic.r.cred"],
742     "if": ["oic.if.baseline"],
743     "p": {"bm": 1},
744     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
745   },
746   {
747     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
748     "href": "/oic/sec/acl2",
749     "rt": ["oic.r.acl2"],
750     "if": ["oic.if.baseline"],
751     "p": {"bm": 1},
752     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
753   },
754   {
755     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
756     "href": "/myFanIntrospection",
757     "rt": ["oic.wk.introspection"],
758     "if": ["oic.if.r", "oic.if.baseline"],
759     "p": {"bm": 3},
760     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
761   },
762   {
763     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
764     "href": "/oic/res",
765     "rt": ["oic.wk.res"],
766     "if": ["oic.if.ll", "oic.if.baseline"],
767     "p": {"bm": 3},
768     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
769   },
770   {
771     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
772     "href": "/oic/d",
773     "rt": ["oic.wk.d", "oic.d.light", "oic.d.virtual"],
774     "if": ["oic.if.r", "oic.if.baseline"],
775     "p": {"bm": 3},
776     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
777   },
778   {
779     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
780     "href": "/oic/p",
781     "rt": ["oic.wk.p"],
782     "if": ["oic.if.r", "oic.if.baseline"],
783     "p": {"bm": 3},
784     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
785   }

```

```

786 },
787 {
788   "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
789   "href": "/myLight",
790   "rt": ["oic.r.switch.binary"],
791   "if": ["oic.if.a", "oic.if.baseline"],
792   "p": {"bm": 3},
793   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
794 },
795 {
796   "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
797   "href": "/oic/sec/doxm",
798   "rt": ["oic.r.doxm"],
799   "if": ["oic.if.baseline"],
800   "p": {"bm": 1},
801   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
802 },
803 {
804   "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
805   "href": "/oic/sec/pstat",
806   "rt": ["oic.r.pstat"],
807   "if": ["oic.if.baseline"],
808   "p": {"bm": 1},
809   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
810 },
811 {
812   "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
813   "href": "/oic/sec/cred",
814   "rt": ["oic.r.cred"],
815   "if": ["oic.if.baseline"],
816   "p": {"bm": 1},
817   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
818 },
819 {
820   "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
821   "href": "/oic/sec/acl2",
822   "rt": ["oic.r.acl2"],
823   "if": ["oic.if.baseline"],
824   "p": {"bm": 1},
825   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
826 },
827 {
828   "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
829   "href": "/myLightIntrospection",
830   "rt": ["oic.wk.introspection"],
831   "if": ["oic.if.r", "oic.if.baseline"],
832   "p": {"bm": 3},
833   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
834 },
835 {
836   "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
837   "href": "/oic/res",
838   "rt": ["oic.wk.res"],
839   "if": ["oic.if.ll", "oic.if.baseline"],
840   "p": {"bm": 3},
841   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
842 },
843 {
844   "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
845   "href": "/oic/d",
846   "rt": ["oic.wk.d", "oic.d.light", "oic.d.virtual"],
847   "if": ["oic.if.r", "oic.if.baseline"],
848 }

```

```

849     "p": {"bm": 3},
850     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
851   },
852   {
853     "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
854     "href": "/oic/p",
855     "rt": ["oic.wk.p"],
856     "if": ["oic.if.r", "oic.if.baseline"],
857     "p": {"bm": 3},
858     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
859   },
860   {
861     "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
862     "href": "/myLight",
863     "rt": ["oic.r.switch.binary"],
864     "if": ["oic.if.a", "oic.if.baseline"],
865     "p": {"bm": 3},
866     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
867   },
868   {
869     "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
870     "href": "/oic/sec/doxm",
871     "rt": ["oic.r.doxm"],
872     "if": ["oic.if.baseline"],
873     "p": {"bm": 1},
874     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
875   },
876   {
877     "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
878     "href": "/oic/sec/pstat",
879     "rt": ["oic.r.pstat"],
880     "if": ["oic.if.baseline"],
881     "p": {"bm": 1},
882     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
883   },
884   {
885     "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
886     "href": "/oic/sec/cred",
887     "rt": ["oic.r.cred"],
888     "if": ["oic.if.baseline"],
889     "p": {"bm": 1},
890     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
891   },
892   {
893     "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
894     "href": "/oic/sec/acl2",
895     "rt": ["oic.r.acl2"],
896     "if": ["oic.if.baseline"],
897     "p": {"bm": 1},
898     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
899   },
900   {
901     "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
902     "href": "/myLightIntrospection",
903     "rt": ["oic.wk.introspection"],
904     "if": ["oic.if.r", "oic.if.baseline"],
905     "p": {"bm": 3},
906     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
907   }
908 ]

```

909 **6.2 General Requirements**

910
911 The translator shall check the protocol-independent UUID (“piid” in OCF) of each device and shall
912 not advertise back into a Bridged Protocol a device originally seen via that Bridged Protocol. The
913 translator shall stop translating any Bridged Protocol device exposed in OCF via another translator
914 if the translator sees the device via the Bridged Protocol. Similarly, the translator shall not
915 advertise an OCF Device back into OCF, and the translator shall stop translating any OCF device
916 exposed in the Bridged Protocol via another translator if the translator sees the device via OCF.
917 These require that the translator can determine when a device is already being translated. A
918 Virtual OCF Device shall be indicated on the OCF network with a Device Type of “oic.d.virtual”.
919 This allows translators to determine if a device is already being translated when multiple translators
920 are present. How a translator determines if a device is already being translated on a non-OCF
921 network is described in the protocol-specific sections below.

922
923 The translator shall detect duplicate virtual devices (with the same protocol-independent UUID)
924 present in a network and shall not create more than one corresponding virtual device as it
925 translates those duplicate devices into another network.

926
927 Each Bridged Server shall be exposed as a separate Virtual OCF Server, with its own endpoint,
928 and its own “/oic/d” and “/oic/p”. The Virtual OCF Server’s “/oic/res” resource would be the same
929 as for any ordinary OCF Server that uses a resource directory. That is, it does not respond to
930 multicast discovery requests (because the OCF Bridge Device responds on its behalf), but a
931 unicast query elicits a response listing its own resources with a “rel”=“hosts” relationship, and an
932 appropriate “anchor” to indicate that it is not the OCF Bridge Device itself. This allows platform-
933 specific, device-specific, and resource-specific fields to all be preserved across translation.

934
935 The introspection data provided by the translator shall include information about all the virtual
936 devices (and their resources) exposed by the translator at that point in time. This means that the
937 introspection data provided by the translator before and after a new virtual device is exposed would
938 be different.

939 **6.3 Security**

940 The OCF Bridge Device shall go through OCF ownership transfer as any other onboardee would.
941 Separately, it shall go through the Bridged Protocol’s ownership transfer mechanism (e.g., AllJoyn
942 claiming) normally as any other onboardee would.

943
944 The OCF Bridge Device shall be field updatable. (This requirement need not be tested but can
945 be certified via a vendor declaration.)

946
947 Unless an administrator opts in to allow it (see section 9.2), a translator shall not expose
948 connectivity to devices that it cannot get a secure connection to.

949 Each Virtual OCF Device shall be provisioned for security by an OCF Onboarding tool. Each Virtual
950 Bridged Device should be provisioned as appropriate in the Bridged ecosystem. In other words,
951 Virtual Devices are treated the same way as physical Devices. They are entities that have to be
952 provisioned in their network.

953 The Translator shall provide a “piid” value that can be used to correlate a non-OCF Device with its
954 corresponding Virtual OCF Device, as specified in Section 6.2. An Onboarding Tool might use this
955 correlation to improve the Onboarding user experience by eliminating or reducing the need for user
956 input, by automatically creating security settings for Virtual OCF Devices that are equivalent to the
957 security settings of their corresponding non-OCF Devices. See the OCF Security Specification for
958 detailed information about Onboarding.

959 Each Virtual Device shall implement the security requirements of the ecosystem that it is connected
960 to. For example, each Virtual OCF Device shall implement the behaviour required by the OCF 1.0
961 Core Specification and the OCF Security Specification. Each Virtual OCF Device shall perform

962 authentication, access control, and encryption according to the security settings it received from
963 the Onboarding Tool.

964 Depending on the architecture of the Translator, authentication and access control might take
965 place just within each ecosystem, but not within the Translator. For example, when an OCF Client
966 sends a request to a Virtual OCF Server:

- 967 - Authentication and access control might be performed by the Virtual OCF Server when
968 receiving the request from the OCF Client.
- 969 - The Translator might not perform authentication or access control when the request travels
970 through the Translator to the corresponding Virtual Bridged Client.
- 971 - Authentication and access control might be performed by the target Bridged Server when
972 it receives the request from the Virtual Bridged Client, according to the security model of
973 the Bridged ecosystem.

974 A Translator may receive unencrypted data coming from a Bridged Client through a Virtual Bridged
975 Device. The translated message shall be encrypted by the corresponding Virtual OCF Client,
976 before sending it to the target OCF Device, if this OCF Device requires encryption.

977 A Translator may receive unencrypted data coming from an OCF Client through a Virtual OCF
978 Server. After translation, this data shall be encrypted by the corresponding Virtual Bridged Client,
979 before sending it to the target Bridged Server, if this Bridged Server requires encryption.

980 A Translator shall protect the data while that data travels between a Virtual Client and a Virtual
981 Server, through the Translator. For example, if the Translator sends data over a network, the
982 Translator shall perform appropriate authentication and access control, and shall encrypt the data,
983 between all peers involved in this communication.

984 **6.3.1 Blocking communication of Bridged Devices with the OCF ecosystem**

985 An OCF Onboarding Tool shall be able to block the communication of all OCF Devices with all
986 Bridged Devices that don't communicate securely with the Bridge, by using the Bridge Device's
987 "oic.r.securemode" Resource.

988 In addition, an OCF Onboarding Tool can block the communication of a particular Virtual OCF
989 Client with all OCF Servers, or block the communication of all OCF Clients with a particular Virtual
990 OCF Server, in the same way as it would for any other OCF Device. See section 8.5 of the OCF
991 Security Specification for information about the soft reset state.

992 **7 AllJoyn Translation**

993 **7.1 Requirements Specific to an AllJoyn Translator**

994 The translator shall be an AllJoyn Router Node. (This is a requirement so that users can expect
995 that a certified OCF Bridge Device will be able to talk to any AllJoyn device, without the user having
996 to buy some other device.)

997 The requirements in this section apply when using algorithmic translation, and by default apply to
998 deep translation unless the relevant specification for such deep translation specifies otherwise.

999 **7.1.1 Exposing AllJoyn producer devices to OCF Clients**

1000 As specified in the OCF Security Specification, the value of the "di" property of OCF Devices
1001 (including Virtual OCF Devices) shall be established as part of Onboarding of that Virtual OCF
1002 Device.

1003
1004 Each AllJoyn object shall be mapped to one or more Virtual OCF Resources. If all AllJoyn
1005 interfaces can be translated to resource types on the same resource (as discussed below), there
1006 should be a single Virtual OCF Resource, and the path component of the URI of the Virtual OCF

1007 Resource shall be the AllJoyn object path, where each “_h” in the AllJoyn object path is
1008 transformed to “-” (hyphen), each “_d” in the AllJoyn object path is transformed to “.” (dot), each
1009 “_t” in the AllJoyn object path is transformed to “~” (tilde), and each “_u” in the AllJoyn object path
1010 is transformed to “_” (underscore). Otherwise, a Resource with that path shall exist with a
1011 Resource Type of [“oic.wk.col”, “oic.r.alljoynobject”] which is a Collection of links, where
1012 “oic.r.alljoynobject” is defined in Section 9.3, and the items in the collection are the Resources with
1013 the translated Resource Types as discussed below.

1014
1015 The value of the “piid” property of “/oic/d” for each Virtual OCF Device shall be the value of the
1016 OCF-defined AllJoyn field “org.openconnectivity.piid” in the AllJoyn About Announce signal, if that
1017 field exists, else it shall be calculated by the Translator as follows:

- 1018 • If the AllJoyn device supports security, the value of the “piid” property value shall be the
1019 peer GUID.
- 1020 • If the AllJoyn device does not support security but the device is being bridged anyway (see
1021 section 9.2), the “piid” property value shall be derived from the Deviceld and Appld
1022 properties (in the About data), by concatenating the Deviceld value (not including any null
1023 termination) and the Appld bytes and using the result as the “name” to be used in the
1024 algorithm specified in IETF RFC 4122 section 4.3, with SHA-1 as the hash algorithm, and
1025 8f0e4e90-79e5-11e6-bdf4-0800200c9a66 as the name space ID. (This is to address the
1026 problem of being able to de-duplicate AllJoyn devices exposed via separate OCF Bridge
1027 Devices.)
1028

1029 A translator implementation is encouraged to listen for AllJoyn About Announce signals matching
1030 any AllJoyn interface name. It can maintain a cache of information it received from these signals,
1031 and use the cache to quickly handle “/oic/res” queries from OCF Clients (without having to wait for
1032 Announce signals while handling the queries).

1033 A translator implementation is encouraged to listen for other signals (including
1034 EmitsChangedSignal of properties) only when there is a client subscribed to a corresponding
1035 resource on a Virtual AllJoyn Device.

1036
1037 There are multiple types of AllJoyn interfaces, which shall be handled as follows.

- 1038 • If the AllJoyn interface is in a well-defined set (defined in OCF ASA Mapping or section
1039 7.1.1.1 below) of interfaces where standard forms exist on both the AllJoyn and OCF
1040 sides, the translator shall either:
1041 a. follow the specification for translating that interface specially, or
1042 b. not translate the AllJoyn interface.
- 1043 • If the AllJoyn interface is not in the well-defined set, the translator shall either:
1044 a. not translate the AllJoyn interface, or
1045 b. algorithmically map the AllJoyn interface as specified in section 7.2 to
1046 custom/vendor-defined Resource Types by converting the AllJoyn interface
1047 name to OCF resource type name(s).
1048
1049

1050 An AllJoyn interface name shall be converted to a Device Type or a set of one or more OCF
1051 Resource Types as follows:

- 1052 1) If the AllJoyn interface has any members, append a suffix “.<seeBelow>” where <seeBelow>
1053 is described below.
- 1054 2) For each upper-case letter present in the entire string, replace it with a hyphen followed by
1055 the lower-case version of that letter (e.g., convert “A” to “-a”).
- 1056 3) If an underscore appears followed by a (lower-case) letter or a hyphen, for each such
1057 occurrence, replace the underscore with two hyphens (e.g., convert “_a” to “--a”, “_-a” to
1058 “---a”).

- 1059 4) For each underscore remaining, replace it with a hyphen (e.g., convert “_1” to “-1”).
- 1060 5) Prepend the “x.” prefix.

1061
 1062 Some examples are shown in the table below. The first three are normal AllJoyn names
 1063 converted to unusual OCF names. The last three are unusual AllJoyn names converted
 1064 (perhaps back) to normal OCF names. (“xn--” is a normal domain name prefix for the
 1065 Punycode-encoded form of an Internationalized Domain Name, and hence can appear in a
 1066 normal vendor-specific OCF name.)
 1067

From AllJoyn name	To OCF name
example.Widget	x.example.-widget
example.my__widget	x.example.my----widget
example.My_Widget	x.example.-my---widget
xn_p1ai.example	x.xn--p1ai.example
xn__90ae.example	x.xn--90ae.example
example.myName_1	x.example.my-name-1

1068
 1069 Each AllJoyn interface that has members and is using algorithmic mapping shall be mapped to one
 1070 or more Resource Types as follows:

- 1071 • AllJoyn Properties with the same EmitsChangedSignal value are mapped to the same
 1072 Resource Type where the value of the <seeBelow> label is the value of
 1073 EmitsChangedSignal. AllJoyn Properties with EmitsChangedSignal values of “const” or
 1074 “false”, are mapped to Resources that are not Observable, whereas AllJoyn Properties with
 1075 EmitsChangedSignal values of “true” or “invalidates” result in Resources that are
 1076 Observable. The Version property in an AllJoyn interface is always considered to have an
 1077 EmitsChangedSignal value of “const”, even if not specified in introspection XML. The name
 1078 of each property on the Resource Type shall be
 1079 “<ResourceType>.<AllJoynPropertyName>”, where each “_d” in the
 1080 <AllJoynPropertyName> is transformed to “.” (dot), and each “_h” in the
 1081 <AllJoynPropertyName> is transformed to “-” (hyphen).
- 1082 • Resource Types mapping AllJoyn Properties with access “readwrite” shall support the
 1083 “oic.if.rw” Interface. Resource Types mapping AllJoyn Properties with access “read” shall
 1084 support the “oic.if.r” Interface. Resource Types supporting both the “oic.if.rw” and “oic.if.r”
 1085 Interfaces shall choose “oic.if.r” as the default Interface.
- 1086 • Each AllJoyn Method is mapped to a separate Resource Type, where the value of the
 1087 <seeBelow> label is the AllJoyn Method name. The Resource Type shall support the
 1088 “oic.if.rw” Interface. Each argument of the AllJoyn Method shall be mapped to a separate
 1089 Property on the Resource Type, where the name of that Property is prefixed with
 1090 “<ResourceType>arg<#>”, where <#> is the 0-indexed position of the argument in the
 1091 AllJoyn introspection xml, in order to help get uniqueness across all Resource Types on
 1092 the same Resource. Therefore, when the AllJoyn argument name is not specified, the
 1093 name of that property is “<ResourceType>arg<#>”, where <#> is the 0-indexed position of
 1094 the argument in the AllJoyn introspection XML. In addition, that Resource Type has an
 1095 extra “<ResourceType>validity” property that indicates whether the rest of the properties
 1096 have valid values. When the values are sent as part of an UPDATE response, the validity
 1097 property is true, and any other properties have valid values. In a RETRIEVE (GET or
 1098 equivalent in the relevant transport binding) response, the validity property is false, and
 1099 any other properties can have meaningless values. If the validity property appears in an
 1100 UPDATE request, its value shall be true (a value of false shall result in an error response).
- 1101 • Each AllJoyn Signal (whether sessionless, sessioncast, or unicast) is mapped to a separate
 1102 Resource Type on an Observable Resource, where the value of the <seeBelow> label is
 1103 the AllJoyn Signal name. The Resource Type shall support the “oic.if.r” Interface. Each

1104 argument of the AllJoyn Signal is mapped to a separate Property on the Resource Type,
 1105 where the name of that Property is prefixed with "<ResourceType>arg<#>", where <#> is
 1106 the 0-indexed position of the argument in the AllJoyn introspection xml, in order to help get
 1107 uniqueness across all Resource Types on the same Resource. Therefore, when the
 1108 AllJoyn argument name is not specified, the name of that property is
 1109 "<ResourceType>arg<#>", where <#> is the 0-indexed position of the argument in the
 1110 AllJoyn introspection XML. In addition, that Resource Type has an extra
 1111 "<ResourceType>validity" property that indicates whether the rest of the properties have
 1112 valid values. When the values are sent as part of a NOTIFY response, the validity property
 1113 is true, and any other properties have valid values. In a RETRIEVE (GET or equivalent in
 1114 the relevant transport binding) response, the validity property is false, and any other
 1115 properties returned can have meaningless values. This is because in AllJoyn, the signals
 1116 are instantaneous events, and the values are not necessarily meaningful beyond the
 1117 lifetime of that message. Note that AllJoyn does have a TTL field that allows store-and-
 1118 forward signals, but such support is not required in OCF 1.0. We expect that in the future,
 1119 the TTL may be used to allow valid values in response to a RETRIEVE that is within the
 1120 TTL.

1121 When an algorithmic mapping is used, AllJoyn data types shall be mapped to OCF property types
 1122 according to Section 7.2.

1123
 1124 If an AllJoyn operation fails, the translator shall send an appropriate OCF error response to the
 1125 OCF client. If an AllJoyn error name is available and does not contain the
 1126 "org.openconnectivity.Error.Code" prefix, it shall construct an appropriate OCF error message (e.g.,
 1127 diagnostic payload if using CoAP) from the AllJoyn error name and AllJoyn error message (if any),
 1128 using the form "<error name>: <error message>", with the <error name> taken from the AllJoyn
 1129 error name field and the <error message> taken from the AllJoyn error message, and the CoAP
 1130 error code set to an appropriate value (if CoAP is used). If an AllJoyn error name is available and
 1131 contains the "org.openconnectivity.Error.Code" prefix, the OCF error message (e.g., diagnostic
 1132 payload if using CoAP) should be taken from the AllJoyn error message (if any), and the CoAP
 1133 error code (if CoAP is used) set to a value derived as follows; remove the
 1134 "org.openconnectivity.Error.Code" prefix, and if the resulting error name is of the form "<#>" where
 1135 <#> is an error code without a decimal (e.g., "404"), the CoAP error code shall be the error code
 1136 indicated by the "<#>". Example: "org.openconnectivity.Error.Code404" becomes "404", which
 1137 shall result in an error 4.04 for a CoAP transport.

1138 **7.1.1.1 Exposing an AllJoyn producer application as a Virtual OCF Server**

1139 Table 1 shows how OCF Device properties, as specified in Table 20 in the OCF 1.0 Core
 1140 Specification, shall be derived, typically from fields specified in the AllJoyn About Interface
 1141 Specification and AllJoyn Configuration Interface Specification.

1142
 1143 If the AllJoyn About or Config data field has a mapping rule defined (as in Table 1, Table 2, Table
 1144 3, and Table 4 below), the field name shall be translated based on that mapping rule; else if the
 1145 AllJoyn About or Config data field has a fully qualified name (with a <domain> prefix (such as
 1146 "com.example", "org.alljoyn"), the field name shall be translated based on the rules specified in
 1147 Section 7.1.1 for mapping AllJoyn fields; else, the field shall not be translated as it may be incorrect
 1148 (error) or it has no valid mapping (such as daemonRealm and passCode).

1149
 1150 **Table 1: oic.wk.d resource type definition**

To OCF Property title	OCF Property name	OCF Description	OCF Mand ?	From AJ Field name	AJ Description	AJ Mand?
(Device) Name	n	Human friendly name	Y	AppName (no exact equivalent exists)	Application name assigned by the app manufacturer	Y

		For example, "Bob's Thermostat"			(developer or the OEM).	
Spec Version	icv	Spec version of the core specification this device is implemented to, The syntax is "core.major.minor"]	Y	(none)	Translator should return its own value	
Device ID	di	Unique identifier for Device. This value shall be as defined in [OCF Security] for DeviceID.	Y	(none)	Use as defined in the OCF Security Specification	
Protocol-Independent ID	piid	Unique identifier for OCF Device (UUID)	Y	org.openconnectivity.piid if it exists, else "Peer GUID" (not in About, but exposed by protocol) if authenticated, else Hash(DeviceId,Appld) where the Hash is done by concatenating the Device Id (not including any null terminator) and the Appld and using the algorithm in IETF RFC 4122 section 4.3, with SHA-1. This means that the value of di may change if the resource is read both before and after authentication, in order to mitigate privacy concerns discussed in RFC 6973.	Peer GUID: The peer GUID is the only persistent identity for a peer. Peer GUIDs are used by the authentication mechanisms to uniquely and identify a remote application instance. The peer GUID for a remote peer is only available if the remote peer has been authenticated. DeviceId: Device identifier set by platform-specific means. Appld: A 128-bit globally unique identifier for the application. The Appld shall be a universally unique identifier as specified in IETF RFC 4122.	Peer GUID: conditionally Y DeviceId: Y Appld: Y
Data Model Version	dmv	Spec version(s) of the vertical specifications this device data model is implemented to. The syntax is a comma separated list of "<vertical>.major.minor"]. <vertical> is the name of the vertical (i.e. sh for Smart Home)	Y	Comma separated list of the Version property values of each interface listed in the objectDescription argument of the Announce signal of About. In addition to the mandatory values specified in the OCF Core specification, additional values are formatted as "x.<interface name>.<Version property value>".	This specification assumes that the value of the Version property is the same as the value of the "org.gtk.GDBus.Since" annotation of the interface in the AllJoyn introspection XML, and therefore the value of the Version property may be determined through introspection alone. Note that AllJoyn specifies that the default value is 1 if the "org.gtk.GDBus.Sin	N, but required by IRB for all standard interfaces, and absence can be used to imply a constant (e.g., 0)

					ce" annotation is absent.	
Localized Descriptions	ld	Detailed description of the Device, in one or more languages. This property is an array of objects where each object has a 'language' field (containing an RFC 5646 language tag) and a 'value' field containing the device description in the indicated language.	N	Description	Detailed description expressed in language tags as in RFC 5646 .	Y
Software Version	sv	Version of the device software.	N	SoftwareVersion	Software version of the app.	Y
Manufacturer Name	dmn	Name of manufacturer of the Device, in one or more languages. This property is an array of objects where each object has a 'language' field (containing an RFC 5646 language tag) and a 'value' field containing the manufacturer name in the indicated language.	N	Manufacturer	The manufacturer's name of the app.	Y
Model Number	dmno	Model number as designated by manufacturer.	N	ModelNumber	The app model number.	Y

1151 In addition, any additional vendor-defined fields in the AllJoyn About data shall be mapped to
 1152 vendor-defined properties in the OCF Device resource "/oic/d" (which implements the "oic.wk.d"
 1153 resource type), with a property name formed by prepending "x." to the AllJoyn field name.
 1154

1155
 1156 Table 2 shows how OCF Device Configuration properties, as specified in Table 15 in the OCF 1.0
 1157 Core Specification, shall be derived:
 1158

1159 **Table 2: oic.wk.con resource type definition**

To OCF Property title	OCF Property name	OCF Description	OCF Mand?	From AJ Field name	AJ Description	AJ Mand?
(Device) Name	n	Human friendly name For example, "Bob's Thermostat"	Y	AppName (no exact equivalent exists)	Application name assigned by the app manufacturer (developer or the OEM).	Y
Location	loc	Provides location information where available.	N	org.openconnectivity.loc (if it exists, else property shall be absent)		N
Location Name	locn	Human friendly name for location For example, "Living Room".	N	org.openconnectivity.locn (if it exists, else property shall be absent)		N
Currency	c	Indicates the currency that is used for any monetary transactions	N	org.openconnectivity.c (if it exists, else property shall be absent)		N
Region	r	Free form text Indicating the current region in	N	org.openconnectivity.r (if it exists, else property shall be absent)		N

		which the device is located geographically. The free form text shall not start with a quote (").				
Localized Names	ln	Human-friendly name of the Device, in one or more languages. This property is an array of objects where each object has a 'language' field (containing an RFC 5646 language tag) and a 'value' field containing the device name in the indicated language. If this property and the Device Name (n) property are both supported, the Device Name (n) value shall be included in this array.	N	AppName	Application name assigned by the app manufacturer (developer or the OEM).	Y
Default Language	dl	The default language supported by the Device, specified as an RFC 5646 language tag. By default, clients can treat any string property as being in this language unless the property specifies otherwise.	N	DefaultLanguage	The default language supported by the device. Specified as an IETF language tag listed in RFC 5646 .	Y

1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170

In addition, any additional vendor-defined fields in the AllJoyn Configuration data shall be mapped to vendor-defined properties in the OCF Configuration resource (which implements the "oic.wk.con" resource type and optionally the "oic.wk.con.p" resource type), with a property name formed by prepending "x." to the AllJoyn field name.

Table 3 shows how OCF Platform properties, as specified in Table 21 in the OCF 1.0 Core Specification, shall be derived, typically from fields specified in the AllJoyn About Interface Specification and AllJoyn Configuration Interface Specification.

Table 3: oic.wk.p Resource Type definition

To OCF Property title	OCF Property name	OCF Description	OCF Mand?	From AJ Field name	AJ Description	AJ Mand?
Platform ID	pi	Unique identifier for the physical platform (UUID); this shall be a UUID in accordance with IETF RFC 4122. It is	Y	DeviceId if it is a UUID, else generate a name-based UUID from the DeviceId using the DeviceId value (not including any null termination) as the "name" to be used in the algorithm specified in	Name of the device set by platform-specific means (such as Linux and Android).	Y

		recommended that the UUID be created using the random generation scheme (version 4 UUID) specific in the RFC.		IETF RFC 4122 section 4.3, with SHA-1 as the hash algorithm, and 8f0e4e90-79e5-11e6-bdf4-0800200c9a66 as the name space ID.		
Manufacturer Name	mnmn	Name of manufacturer (not to exceed 16 characters)	Y	Manufacturer (in DefaultLanguage, truncated to 16 characters)	The manufacturer's name of the app.	Y
Manufacturer Details Link (URL)	mnmml	URL to manufacturer (not to exceed 32 characters)	N	org.openconnectivity.mnmml (if it exists, else property shall be absent)		N
Model Number	mnmno	Model number as designated by manufacturer	N	ModelNumber	The app model number.	Y
Date of Manufacture	mnmndt	Manufacturing date of device	N	DateOfManufacture	Date of manufacture using format YYYY-MM-DD (known as XML DateTime format).	N
Platform Version	mnmnpv	Version of platform – string (defined by manufacturer)	N	org.openconnectivity.mnmnpv (if it exists, else property shall be absent)		N
OS Version	mnmnos	Version of platform resident OS – string (defined by manufacturer)	N	org.openconnectivity.mnmnos (if it exists, else property shall be absent)		N
Hardware Version	mnmnhw	Version of platform hardware	N	HardwareVersion	Hardware version of the device on which the app is running.	N
Firmware version	mnmfv	Version of device firmware	N	org.openconnectivity.mnmfv (if it exists, else property shall be absent)		N
Support URL	mnmnsurl	URL that points to support information from manufacturer	N	SupportUrl	Support URL (populated by the manufacturer)	N
SystemTime	st	Reference time for the device	N	org.openconnectivity.st (if it exists, else property shall be absent)		N
Vendor ID	vid	Vendor defined string for the platform. The string is freeform and up to the vendor on what text to populate it.	N	DeviceId	Name of the device set by platform-specific means (such as Linux and Android).	Y

1171
1172
1173

Table 4 shows how OCF Platform Configuration properties, as specified in Table 16 in the OCF 1.0 Core Specification, shall be derived:

1174
1175

Table 4: oic.wk.con.p Resource Type definition

To OCF Property title	OCF Property name	OCF Description	OCF Mand?	From AJ Field name	AJ Description	AJ Mand?
Platform Names	Mnpn	Platform Identifier	N	DeviceName	Name of the device set by platform-specific means (such as Linux and Android).	Device name assigned by the user. The device name appears on the UI as the friendly name of the device.

1176
1177
1178

In addition, the “oic.wk.mnt” properties Factory_Reset (“fr”) and Reboot (“rb”) shall be mapped to AllJoyn Configuration methods FactoryReset and Restart, respectively.

1179 **7.1.2 Exposing OCF resources to AllJoyn consumer applications**

1180 Unless specified otherwise, each OCF resource shall be mapped to a separate AllJoyn object.

1181

1182 Each OCF Server shall be exposed as a separate AllJoyn producer application, with its own About
1183 data. This allows platform-specific, device-specific, and resource-specific fields to all be
1184 preserved across translation. However, this requires that AllJoyn Claiming of such producer
1185 applications be solved in a way that does not require user interaction, but this is left as an
1186 implementation issue.

1187

1188 The AllJoyn producer application shall implement the “oic.d.virtual” AllJoyn interface. This allows
1189 translators to determine if a device is already being translated when multiple translators are
1190 present. The “oic.d.virtual” interface is defined as follows:

1191

```
1192 <interface name="oic.d.virtual"/>
```

1193

1194 The implementation may choose to implement this interface by the AllJoyn object at path “/oic/d”.

1195

1196 The AllJoyn peer ID shall be the OCF device ID (“di”).

1197

1198 Unless specified otherwise, the AllJoyn object path shall be the OCF URI path, where each “-”
1199 (hyphen) in the OCF URI path is transformed to “_h”, each “.” (dot) in the OCF URI path is
1200 transformed to “_d”, each “~” (tilde) in the OCF URI path is transformed to “_t”, and each “_”
1201 (underscore) in the OCF URI path is transformed to “_u”.

1202

1203 The AllJoyn About data shall be populated per Table 5 below.

1204

1205 A translator implementation is encouraged to maintain a cache of OCF resources to handle
1206 WholImplements queries from the AllJoyn side, and emit an Announce Signal for each OCF Server.
1207 Specifically, the translator could always Observe “/oic/res” changes and only Observe other
1208 resources when there is a client with a session on a Virtual AllJoyn Device.

1209

1210 There are multiple types of resources, which shall be handled as follows.

1211

- If the Resource Type is in a well-defined set (defined in OCF ASA Mapping or section 7.1.2.1 below) of resource types where standard forms exist on both the AllJoyn and OCF sides, the translator shall either:

1212

1213

- a. follow the specification for translating that resource type specially, or

1214

- 1215 b. not translate the Resource Type.
- 1216 • If the Resource Type is not in the well-defined set (but is not a Device Type), the translator
- 1217 shall either:
- 1218 a. not translate the Resource Type, or
- 1219 b. algorithmically map the Resource Type as specified in section 7.2 to a
- 1220 custom/vendor-defined AllJoyn interface by converting the OCF Resource Type
- 1221 name to an AllJoyn Interface name.
- 1222

1223 An OCF Resource Type or Device Type shall be converted to an AllJoyn interface name as follows:

- 1224 1) Remove the “x.” prefix if present
- 1225 2) For each occurrence of a hyphen (in order from left to right in the string):
- 1226 a. If the hyphen is followed by a letter, replace both characters with a single upper-
- 1227 case version of that letter (e.g., convert “-a” to “A”).
- 1228 b. Else, if the hyphen is followed by another hyphen followed by either a letter or a
- 1229 hyphen, replace two hyphens with a single underscore (e.g., convert “--a” to “_a”,
- 1230 “---” to “_”).
- 1231 c. Else, convert the hyphen to an underscore (i.e., convert “-” to “_”).
- 1232

1233 Some examples are shown in the table below. The first three are unusual OCF names

1234 converted (perhaps back) to normal AllJoyn names. The last three are normal OCF names

1235 converted to unusual AllJoyn names. (“xn--” is a normal domain name prefix for the Punycode-

1236 encoded form of an Internationalized Domain Name, and hence can appear in a normal vendor-

1237 specific OCF name.)

1238

From OCF name	To AllJoyn name
x.example.-widget	example.Widget
x.example.my----widget	example.my__widget
x.example.-my---widget	example.My_Widget
x.xn--p1ai.example	xn_p1ai.example
x.xn--90ae.example	xn__90ae.example
x.example.my-name-1	example.myName_1

1239 An OCF Device Type is mapped to an AllJoyn interface with no members.

1240 Unless specified otherwise, each OCF Resource Type shall be mapped to an AllJoyn interface as

1241 follows:

- 1242 • Each OCF property is mapped to an AllJoyn property in that interface, where each “.” (dot)
- 1243 in the OCF property is transformed to “_d”, and each “-” (hyphen) in the OCF property is
- 1244 transformed to “_h”.
- 1245 • The EmitsChangedSignal value for each AllJoyn property shall be set to “true” if the
- 1246 resource supports NOTIFY, or “false” if it does not. (The value is never set to “const” or
- 1247 “invalidates” since those concepts cannot currently be expressed in OCF.)
- 1248 • The “access” attribute for each AllJoyn property shall be “read” if the OCF property is read-
- 1249 only, or “readwrite” if the OCF property is read-write.
- 1250 • If the resource supports DELETE, a Delete() method shall appear in the interface.
- 1251 • If the resource supports CREATE, a Create() method shall appear in the interface, with
- 1252 input arguments of each property of the resource to create. (Such information is not
- 1253 available algorithmically in OIC 1.1 but can be determined in OCF 1.0 via introspection.) If
- 1254 such information is not available, a CreateWithDefaultValues() method shall appear which
- 1255 takes no input arguments. In either case, the output argument shall be an OBJECT_PATH
- 1256 containing the path of the created resource.
- 1257
- 1258

- If the resource supports UPDATE (i.e., the “oic.if.rw” or “oic.if.a” interface) then an AllJoyn property set operation (i.e., an org.freedesktop.DBus.Properties.Set() method call) shall be mapped to a Partial UPDATE (e.g., POST in CoAP) with the corresponding OCF property.
- If a Resource has a Resource Type “oic.r.alljoynobject”, then instead of separately translating each of the Resources in the collection to its own AllJoyn object, all Resources in the collection shall instead be translated to a single AllJoyn object whose object path is the OCF URI path of the collection.

OCF property types shall be mapped to AllJoyn data types according to Section 7.2.

If an OCF operation fails, the translator shall send an appropriate AllJoyn error response to the AllJoyn consumer. If an error message is present in the OCF response, and the error message (e.g., diagnostic payload if using CoAP) fits the pattern "<error name>: <error message>" where <error name> conforms to the AllJoyn error name syntax requirements, the AllJoyn error name and AllJoyn error message shall be extracted from the error message in the OCF response. Otherwise, the AllJoyn error name shall be "org.openconnectivity.Error.Code<#>" where <#> is the error code (e.g., CoAP error code) in the OCF response without a decimal (e.g., "404") and the AllJoyn error message is the error message in the OCF response.

7.1.2.1 Exposing an OCF server as a Virtual AllJoyn Producer

The object description returned in the About interface shall be formed as specified in the AllJoyn About Interface Specification, and Table 5 shows how AllJoyn About Interface fields shall be derived, based on properties in “oic.wk.d”, “oic.wk.con”, “oic.wk.p”, and “oic.wk.con.p”.

Table 5: AllJoyn About Data fields

To AJ Field name	AJ Description	AJ Mand?	From OCF Property title	OCF Property name	OCF Description	OCF Mand?
AppId	A 128-bit globally unique identifier for the application. The AppId shall be a universally unique identifier as specified in RFC 4122 .	Y	Device ID (no exact equivalent exists)	di	Unique identifier for OCF Device (UUID)	Y
DefaultLanguage	The default language supported by the device. Specified as an IETF language tag listed in RFC 5646 .	Y	Default Language	dl	The default language supported by the Device, specified as an RFC 5646 language tag. By default, clients can treat any string property as being in this language unless the property specifies otherwise. If absent, the translator shall return a constant, e.g., empty string	N
DeviceName (per supported language)	Name of the device set by platform-specific means (such as Linux and Android).	N	Platform Names	mnpn	Friendly name of the Platform. This property is an array of objects where each object has a 'language' field (containing an RFC 5646 language tag) and a 'value' field containing the platform friendly name in the indicated language.	N

To AJ Field name	AJ Description	AJ Mand?	From OCF Property title	OCF Property name	OCF Description	OCF Mand?
					For example, [{"language": "en", "value": "Dave's Laptop"}]	
DeviceId	Device identifier set by platform-specific means.	Y	Platform ID	pi	Platform Identifier	Y
AppName (per supported language)	Application name assigned by the app manufacturer (developer or the OEM).	Y	Localized Names, if it exists, else (Device) Name	ln or n	Human-friendly name of the Device, in one or more languages. This property is an array of objects where each object has a 'language' field (containing an RFC 5646 language tag) and a 'value' field containing the device name in the indicated language. If this property and the Device Name (n) property are both supported, the Device Name (n) value shall be included in this array.	N (ln), Y (n)
Manufacturer (per supported language)	The manufacturer's name of the app.	Y	Manufacturer Name	dmn	Name of manufacturer of the Device, in one or more languages. This property is an array of objects where each object has a 'language' field (containing an RFC 5646 language tag) and a 'value' field containing the manufacturer name in the indicated language.	N
ModelNumber	The app model number.	Y	Model Number	dmno	Model number as designated by manufacturer	N
SupportedLanguages	List of supported languages.	Y	language fields of Localized Names	ln	If ln is supported, return the list of values of the language field of each array element, else return empty array	N
Description (per supported language)	Detailed description expressed in language tags as in RFC 5646 .	Y	Localized Descriptions	ld	Detailed description of the Device, in one or more languages. This property is an array of objects where each object has a 'language' field (containing an RFC 5646 language tag) and a 'value' field containing the device description in the indicated language.	N
DateOfManufacture	Date of manufacture using format YYYY-MM-DD (known as XML DateTime format).	N	Date of Manufacture	mndt	Manufacturing date of device	N
SoftwareVersion	Software version of the app.	Y	Software Version	sv	Software version of the device.	N
AJSoftwareVersion	Current version of the AllJoyn SDK	Y	(none)		Translator should return its own value	

To AJ Field name	AJ Description	AJ Mand?	From OCF Property title	OCF Property name	OCF Description	OCF Mand?
	used by the application.					
HardwareVersion	Hardware version of the device on which the app is running.	N	Hardware Version	mnhw	Version of platform hardware	N
SupportUrl	Support URL (populated by the manufacturer).	N	Support URL	mnsl	URL that points to support information from manufacturer	N
org.openconnectivity.mnml		N	Manufacturer Details Link (URL)	mnml (if it exists, else field shall be absent)	URL to manufacturer (not to exceed 32 characters)	N
org.openconnectivity.mnpv		N	Platform Version	mnpv (if it exists, else field shall be absent)	Version of platform – string (defined by manufacturer)	N
org.openconnectivity.mnos		N	OS Version	mnos (if it exists, else field shall be absent)	Version of platform resident OS – string (defined by manufacturer)	N
org.openconnectivity.mnfv		N	Firmware version	mnfv (if it exists, else field shall be absent)	Version of device firmware	N
org.openconnectivity.st		N	SystemTime	st (if it exists, else field shall be absent)	Reference time for the device	N
org.openconnectivity.piid		N	Protocol-Independent ID	piid	A unique and immutable Device identifier. A Client can detect that a single Device supports multiple communication protocols if it discovers that the Device uses a single Protocol Independent ID value for all the protocols it supports.	Y

1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296

The AllJoyn field “org.openconnectivity.piid” shall be announced but shall not be localized and its D-Bus type signature shall be “s”. All other AllJoyn field names listed in Table 5 which have the prefix “org.openconnectivity.” shall be neither announced nor localized and their D-Bus type signature shall be “s”.

In addition, any additional vendor-defined properties in the OCF Device resource “/oic/d” (which implements the “oic.wk.d” resource type) and the OCF Platform resource “/oic/p” (which implements the “oic.wk.p” resource type) shall be mapped to vendor-defined fields in the AllJoyn About data, with a field name formed by removing the leading “x.” from the property name.

Table 6 shows how AllJoyn Configuration Interface fields shall be derived, based on properties in “oic.wk.con” and “oic.wk.con.p”.

Table 6: AllJoyn Configuration Data fields

To AJ Field name	AJ Description	AJ Mand?	From OCF Property title	OCF Property name	OCF Description	OCF Mand?
DefaultLanguage	Default language supported by the device.	N	Default Language	dl	The default language supported by the Device, specified as an RFC 5646 language tag. By default, clients can treat any string property as being in this language unless the property specifies otherwise.	N
DeviceName	Device name assigned by the user. The device name appears on the UI as the friendly name of the device.	N	PlatformNames	mnpn	Friendly name of the Platform. This property is an array of objects where each object has a 'language' field (containing an RFC 5646 language tag) and a 'value' field containing the platform friendly name in the indicated language. For example, [{"language": "en", "value": "Dave's Laptop"}]	N
org.openconnectivity.loc		N	Location	loc (if it exists, else field shall be absent)	Provides location information where available.	N
org.openconnectivity.locn		N	Location Name	locn (if it exists, else field shall be absent)	Human friendly name for location For example, "Living Room".	N
org.openconnectivity.c		N	Currency	c (if it exists, else field shall be absent)	Indicates the currency that is used for any monetary transactions	N
org.openconnectivity.r		N	Region	r (if it exists, else field shall be absent)	Free form text Indicating the current region in which the device is located geographically. The free form text shall not start with a quote (").	N

1298

1299 The AllJoyn field "org.openconnectivity.loc" shall be neither announced nor localized and its D-
1300 Bus type signature shall be "ad". All other AllJoyn field names listed in Table 5 which have the
1301 prefix "org.openconnectivity." shall be neither announced nor localized and their D-Bus type
1302 signature shall be "s".

1303

1304 In addition, the Configuration methods FactoryReset and Restart shall be mapped to "oic.wk.mnt"
1305 properties Factory_Reset ("fr") and Reboot ("rb"), respectively, and any additional vendor-defined
1306 properties in the OCF Configuration resource (which implements the "oic.wk.con" resource type
1307 and optionally the "oic.wk.con.p" resource type) shall be mapped to vendor-defined fields the
1308 AllJoyn Configuration data, with a field name formed by removing the leading "x." from the property
1309 name.

1310

1311 **7.2 On-the-Fly Translation from D-Bus and OCF payloads**

1312 The “dbus1” payload format is specified in the D-Bus Specification and AllJoyn adopted the D-Bus
1313 protocol and made it distributed over the network. The modifications done by AllJoyn to the format are all
1314 in the header part of the packet, not in the data payload itself, which remains compatible with “dbus1”. Other
1315 variants of the protocol that have been proposed by the Linux community (“GVariant” and “kdbus” payloads)
1316 contain slight incompatibilities and are not relevant for this discussion.

1317 **7.2.1 Translation without aid of introspection**

1318 This section describes how translators shall translate messages between the two payload formats in the
1319 absence of introspection metadata from the actual device. This situation arises in the following cases:

- 1320 • Requests to OIC 1.1 devices
- 1321 • Replies from OIC 1.1 devices
- 1322 • Content not described by introspection, such as the inner payload of AllJoyn properties of type
1323 “D-Bus VARIANT”.

1324 Since introspection is not available, the translator cannot know the rich JSON sub-type, only the underlying
1325 CBOR type and from that it can infer the JSON generic type, and hence translation is specified below in
1326 terms of those generic types.

1327 **7.2.1.1 Booleans**

1328 Boolean conversion is trivial since both sides support this type.

D-Bus type	JSON type
“b” – BOOLEAN	boolean (true or false)

1329 **7.2.1.2 Numeric types**

1330 The translation of numeric types is lossy and that is unavoidable due to the limited expressiveness
1331 of the JSON generic types. This can only be solved with introspection.

1332 The translation of numeric types is direction-specific.

From D-Bus type	To JSON type
“y” - BYTE (unsigned 8-bit)	number
“n” - UINT16 (unsigned 16-bit)	
“u” - UINT32 (unsigned 32-bit)	
“t” - UINT64 (unsigned 64-bit) ⁽¹⁾	
“q” - INT16 (signed 16-bit)	
“” - INT32 (signed 32-bit)	
“x” - INT64 (signed 64-bit) ⁽¹⁾	
“d” - DOUBLE (IEEE 754 double precision)	

1333

From JSON type	To D-Bus type
number	"d" - DOUBLE ⁽²⁾

1334

1335 Notes and rationales:

- 1336 1. D-Bus payloads of types "t" (UINT64) and "x" (INT64) can contain values that cannot be perfectly
 1337 represented in IEEE 754 double-precision floating point. The RFCs governing JSON do not forbid
 1338 such numbers but caution that many implementations may not be able to deal with them.
 1339 Currently, OCF transports its payload using CBOR instead of JSON, which can represent those
 1340 numbers with fidelity. However, it should be noted that the OCF 1.0 Core Specification does
 1341 not allow for integral numbers outside the range $-2^{53} \leq x \leq 2^{53}$.
- 1342 2. To provide the most predictable result, all translations from OCF to AllJoyn produce values of
 1343 type "d" DOUBLE (IEEE 754 double precision).

1344 **7.2.1.3 Text strings**

D-Bus type	JSON type
"s" – STRING	string

1345

1346 Conversion between D-Bus and JSON strings is simple, as both require their content to be valid Unicode.
 1347 For example, an implementation can typically do a direct byte copy, as both protocols specify UTF-8 as
 1348 the encoding of the data, neither constrains the data to a given normalisation format nor specify whether
 1349 private-use characters or non-characters should be disallowed.

1350 Since the length of D-Bus strings is always known, it is recommended translators not use CBOR
 1351 indeterminate text strings (first byte 0x7f).

1352 **7.2.1.4 Byte arrays**

1353 The translation of a byte array is direction-specific.

From D-Bus type	To JSON type
"ay" - ARRAY of BYTE	(base64-encoded) string

1354

1355 The base64url encoding is specified in IETF RFC 4648 section 5.

1356 **7.2.1.5 D-Bus Variants**

D-Bus type	JSON type
"v" – VARIANT	<i>see below</i>

1357

1358 D-Bus has a type called VARIANT ("v") that is a wrapper around any other D-Bus type. It's a way for the
 1359 type system to perform type-erasure. JSON, on the other hand, is not type-safe, which means that all
 1360 JSON values are, technically, variants. The conversion for a D-Bus variant to JSON is performed by
 1361 entering that variant and encoding the type carried inside as per the rules in this document.

1362 The algorithm must be recursive, as D-Bus variants are allowed to contain variants themselves.

1363 7.2.1.6 D-Bus Object Paths and Signatures

1364 The translation of D-Bus object paths and signatures is unidirectional (there is no mapping to them,
1365 only *from* them). In the reverse direction, Section 7.2.1.3 always converts to D-Bus STRING
1366 rather than OBJECT_PATH or SIGNATURE since it is assumed that “s” is the most common string
1367 type in use.

From D-Bus type	To JSON type
“o” - OBJECT_PATH	string
“g” – SIGNATURE	

1368

1369 Both D-Bus object paths and D-Bus type signatures are US-ASCII strings with specific formation rules,
1370 found in the D-Bus Specification. They are very seldom used and are not expected to be found in
1371 resources subject to translation without the aid of introspection.

1372 7.2.1.7 D-Bus Structures

1373 The translation of the following types is direction-specific:

From D-Bus type	To JSON type
“r” – STRUCT	array, length > 0

1374

1375 D-Bus structures can be interpreted as a fixed-length array containing a pre-determined list of types for
1376 each member. This is how such a structure is mapped to JSON: as an array of heterogeneous content,
1377 which are the exact members of the D-Bus structure, in the order in which they appear in the structure.

1378 7.2.1.8 Arrays

1379 The translation of the following types is bidirectional:

D-Bus type	JSON type
“ay” - ARRAY of BYTE	(base64-encoded) string – see Section 7.2.1.4
“ae” - ARRAY of DICT_ENTRY	object – see Section 7.2.1.9

1380

1381 The translation of the following types is direction-specific:

From D-Bus type	To JSON type
“a” – ARRAY of anything else not specified above	array

1382

1383

From JSON type	Condition	To D-Bus type
array	length=0	“av” – ARRAY of VARIANT
array	length>0, all elements of same type	“a” – ARRAY
array	length>0, elements of different types	“r” – STRUCT

1384

1385 Aside from arrays of bytes and arrays of dictionary entries, which are mapped to JSON strings and objects
 1386 respectively, arrays in JSON cannot be constrained to a single type (i.e., heterogeneous arrays). For that
 1387 reason, strictly speaking all D-Bus arrays excepting arrays of bytes and arrays of dictionary entries must
 1388 first be converted to arrays of variant “av” and then that array can be converted to JSON.

1389 Conversion of D-Bus arrays of variants uses the conversion of variants as specified above, which simply
 1390 eliminates the distinction between a variant containing a given value and that value outside a variant. In
 1391 other words, the elements of a D-Bus array are extracted and sent as elements of the JSON array, as per
 1392 the other rules of this document.

1393 **7.2.1.9 Dictionaries / Objects**

D-Bus type	JSON type
“a{sv}” - dictionary of STRING to VARIANT	object

1394

1395 The choice of “dictionary of STRING to VARIANT” is made because that is the most common type of
 1396 dictionary found in payloads and is an almost perfect superset of all possible dictionaries in D-Bus anyway.
 1397 Moreover, it can represent JSON Objects with fidelity, which is the representation that OCF uses in its data
 1398 models, which in turn means those D-Bus dictionaries will be able to carry with fidelity any OCF JSON
 1399 Object in current use.

1400 D-Bus dictionaries that are not mapping string to variant are first converted to those constraints and then
 1401 encoded in CBOR.

1402 **7.2.1.10 Non-translatable types**

D-Bus Type	JSON type
“h” – UNIX_FD (Unix file descriptor)	null
	undefined (not officially valid JSON, but some implementations permit it)

1403

1404 The above types are not translatable, and the translator should drop the incoming message. None of the
 1405 types above are in current use by either AllJoyn, OIC 1.1, or future OCF 1.0 devices, so the inability to
 1406 translate them should not be a problem.

1407 **7.2.1.11 Examples**

1408

Source D-Bus	JSON Result
BOOLEAN(FALSE)	false
BOOLEAN(TRUE)	true
VARIANT(BOOLEAN(FALSE))	false
VARIANT(BOOLEAN(TRUE))	true
BYTE(0)	0.0
BYTE(255)	255.0
INT16(0)	0.0
INT16(-1)	-1.0
INT16(-32768)	-32768.0
UINT16(0)	0.0
UINT16(65535)	65535.0
INT32(0)	0.0
INT32(-2147483648)	-2147483648.0
INT32(2147483647)	2147483647.0
UINT32(0)	0.0
UINT32(4294967295)	4294967295.0
INT64(0)	0.0
INT64(-1)	-1.0
UINT64(18446744073709551615)	18446744073709551615.0 ⁽¹⁾
DOUBLE(0.0)	0.0
DOUBLE(0.5)	0.5
STRING("")	""
STRING("Hello")	"Hello"
ARRAY<BYTE>()	""
ARRAY<BYTE>(0x48, 0x65, 0x6c, 0x6c, 0x6f)	"SGVsbG8"
OBJECT_PATH("/")	"/"
SIGNATURE()	""
SIGNATURE("s")	"s"
VARIANT(INT32(0))	0
VARIANT(VARIANT(INT32(0)))	0
VARIANT(STRING("Hello"))	"Hello"

Source JSON	D-Bus Result
false	BOOLEAN(false)
true	BOOLEAN(true)
0	DOUBLE(0.0)
-1	DOUBLE(-1.0)
-2147483648	DOUBLE(-2147483648.0)
2147483647	DOUBLE(2147483647.0)
2147483648	DOUBLE(2147483648.0)
-2147483649	DOUBLE(-2147483649.0)
9223372036854775808 ⁽¹⁾	DOUBLE(9223372036854775808.0)
0.0	DOUBLE(0.0)
0.5	DOUBLE(0.5)
0.0f	DOUBLE(0.0)
0.5f	DOUBLE(0.5)
""	STRING("")
"Hello"	STRING("Hello")
[]	ARRAY<VARIANT>()
[1]	ARRAY<DOUBLE>(DOUBLE(1.0))
[1, 2147483648, false, "Hello"]	STRUCT<DOUBLE, DOUBLE, BOOLEAN, STRING>(DOUBLE(1.0), DOUBLE(2147483648.0), BOOLEAN(false), STRING("Hello"))
{}	map<STRING, VARIANT>()
{1: 1}	map<STRING, VARIANT>("1" → VARIANT(DOUBLE(1.0)))
{"1": 1}	map<STRING, VARIANT>("1" → VARIANT(DOUBLE(1.0)))
{ "rep": { "state": false, "power": 1.0, "name": "My Light" } }	map<STRING, VARIANT>({STRING("rep"), VARIANT(map<STRING, VARIANT>({STRING("state") → VARIANT(BOOLEAN(FALSE))}, {STRING("power") → VARIANT(DOUBLE(1.0))}, {STRING("name") → VARIANT(STRING("My Light"))}))})

1411
 1412
 1413
 1414
 1415

Note:

1. This value cannot be represented with IEEE754 double-precision floating point without loss of information. It is also outside the currently-allowed range of integrals in OCF.

1416 **7.2.2 Translation with aid of introspection**

1417 When introspection is available, the translator can use the extra metadata provided by the side offering the
1418 service to expose a higher-quality reply to the other side. This chapter details modifications to the translation
1419 described in the previous chapter when the metadata is found.

1420 Introspection metadata can be used for both translating requests to services and replies from those services.
1421 When used to translate requests, the introspection is “constraining”, since the translator must conform
1422 exactly to what that service expects. When used to translate replies, the introspection is “relaxing”, but may
1423 be used to inform the receiver what other possible values may be encountered in the future.

1424 Note that OCF introspection uses JSON types, media attributes, and format attributes, not CBOR encoding.
1425 The actual encoding of each JSON type is discussed in Section 12.3 of the OCF 1.0 Core Specification,
1426 JSON format attribute values are as defined in JSON Schema Validation, and JSON media attribute
1427 values are as defined in JSON Hyper-Schema.

1428 **7.2.2.1 Translation of the introspection itself**

1429 Note that both OCF 1.0 and AllJoyn require all services exposed to include introspection metadata, which
1430 means the translator will need to translate the introspection information on-the-fly for each OCF resource
1431 or AllJoyn producer it finds. The translator shall preserve as much of the original information as can be
1432 represented in the translated format. This includes both the information used in machine interactions and
1433 the information used in user interactions, such as description and documentation text.

1434 **7.2.2.2 Variability of introspection data**

1435 Introspection data is not a constant and the translator may find, upon discovering further services, that the
1436 D-Bus interface or OCF Resource Type it had previously encountered is different than previously seen. The
1437 translator needs to take care about how the destination side will react to a change in introspection.

1438 D-Bus interfaces used by AllJoyn services may be updated to newer versions, which means a given type
1439 of service may be offered by two distinct versions of the same interface. Updates to standardised interfaces
1440 must follow strict guidelines established by the AllSeen Interface Review Board, mapping each version to
1441 a different OCF Resource Type should be possible without much difficulty. However, there’s no guarantee
1442 that vendor-specific extensions follow those requirements. Indeed, there’s nothing preventing two revisions
1443 of a product to contain completely incompatible interfaces that have the same name and version number.

1444 On the opposite direction, the rules are much laxer. Since OCF specifies optional properties to its Resource
1445 Types, a simple monotonically-increasing version number like AllJoyn consumer applications expect is not
1446 possible.

1447 However, it should be noted that services created by the translator by “on-the-fly” translation will only be
1448 accessed by generic client applications. Dedicated applications will only use “deep binding” translation.

1449 **7.2.2.3 Numeric types**

1450 For numeric values, all D-Bus and JSON numeric types are treated equally as source and may all be
1451 translated into any of the other side’s types. When translating a request to a service, the translator need
1452 only verify whether there would be loss of information when translating from source to destination. For
1453 example, when translating the number 1.5 to either a JSON integer or to one of the D-Bus integral types,
1454 there would be loss of information, in which case the translator should refuse the incoming message.
1455 Similarly, the value 1,234,567 does not fit the range of a D-Bus byte, 16-bit signed or unsigned integer.

1456 When translating the reply from the service, the translator shall use the following rules.

1457 The following table indicates how to translate from a JSON type to the corresponding D-Bus type, where
 1458 the first matching row shall be used. If the JSON schema does not indicate the minimum value of a JSON
 1459 integer, 0 is the default. If the JSON schema does not indicate the maximum value of a JSON integer, 2^{32}
 1460 – 1 is the default. The resulting AllJoyn introspection XML shall contain “org.alljoyn.Bus.Type.Min” and
 1461 “org.alljoyn.Bus.Type.Max” annotations whenever the minimum or maximum, respectively, of the JSON
 1462 value is different from the natural minimum or maximum of the D-Bus type.

From JSON type	Condition	To D-Bus Type
integer	minimum ≥ 0 AND maximum $< 2^8$	“y” (BYTE)
	minimum ≥ 0 AND maximum $< 2^{16}$	“q” (UINT16)
	minimum $\geq -2^{15}$ AND maximum $< 2^{15}$	“n” (INT16)
	minimum ≥ 0 AND maximum $< 2^{32}$	“u” (UINT32)
	minimum $\geq -2^{31}$ AND maximum $< 2^{31}$	“i” (INT32)
	minimum ≥ 0	“t” (UINT64)
		“x” (INT64)
number		“d” (DOUBLE)
string	pattern = “ $^0 ([1-9][0-9]{0,19})\$$ ”	“t” (UINT64)
	pattern = “ $^0 (-?[1-9][0-9]{0,18})\$$ ”	“x” (INT64)

1463

1464 The following table indicates how to translate from a D-Bus type to the corresponding JSON type.

From D-Bus type	To JSON type	Note
“y” (BYTE)	integer	“minimum” and “maximum” in the JSON schema shall be set to the value of the “org.alljoyn.Bus.Type.Min” and “org.alljoyn.Bus.Type.Max” (respectively) annotations if present, or to the min and max values of the D-Bus type’s range if such annotations are absent.
“n” (UINT16)		
“q” (INT16)		
“u” (UINT32)		
“i” (INT32)		
“t” (UINT64)	integer if org.alljoyn.Bus.Type.Max $\leq 2^{53}$, else string with JSON pattern attribute “ $^0 ([1-9][0-9]{0,19})\$$ ”.	IETF RFC 7159 section 6 explains that higher JSON integers are not interoperable.
“x” (INT64)	integer (if org.alljoyn.Bus.Type.Min $\geq -2^{53}$ AND org.alljoyn.Bus.Type.Max $\leq 2^{53}$), else string with JSON pattern attribute “ $^0 (-?[1-9][0-9]{0,18})\$$ ”.	IETF RFC 7159 section 6 explains that other JSON integers are not interoperable.
“d” (double)	number	

1465

1466 **7.2.2.4 Text string and byte arrays**

D-Bus Type	JSON type	JSON media attribute, binaryEncoding property
"s" – STRING	string	(none)
"ay" - ARRAY of BYTE	string	base64

1467

1468 There's no difference in the translation of text strings and byte arrays compared to the previous section.
 1469 This section simply lists the JSON equivalent types for the generated OCF introspection.

1470 In addition, the mapping of the following JSON Types is direction-specific:

From JSON type	Condition	To D-Bus Type
string	pattern = <code>^[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{12}\$</code>	"ay" – ARRAY of BYTE

1471

1472 JSON strings with any other format value (e.g., date-time, uri, etc.) or pattern value not shown in this table
 1473 above shall be treated the same as if the format and pattern attributes were absent, by simply mapping
 1474 the value to a D-Bus string.

1475 **7.2.2.5 D-Bus Variants**

D-Bus Type	JSON Type
"v" – VARIANT	<i>see below</i>

1476

1477 If the introspection of an AllJoyn producer indicates a value in a request should be a D-Bus VARIANT, the
 1478 translator should create such a variant and encode the incoming value as the variant's payload as per the
 1479 rules in the rest of this document.

1480 **7.2.2.6 D-Bus Object Paths and Signatures**

From D-Bus Type	To JSON Type
"o" – OBJECT_PATH	string
"g" – SIGNATURE	

1481

1482 If the introspection of an AllJoyn producer indicates a value in a request should be a D-Bus Object Path or
 1483 D-Bus Signature, the translator should perform a validity check in the incoming CBOR Text String. If the
 1484 incoming data fails to pass this check, the message should be rejected.

1485 **7.2.2.7 D-Bus Structures**

1486 D-Bus structure members are described in the introspection XML with the
 1487 "org.alljoyn.Bus.Struct.*StructureName*.Field.*fieldName*.Type" annotation. The translator shall use
 1488 the AJSoftwareVersion field of the About data obtained from a bridged AllJoyn producer as follows.

1489 When the version of AllJoyn implemented on the Bridged Device is v16.10.00 or greater and the
 1490 member annotations are present, the translator shall use a JSON object to represent a structure,
 1491 mapping each member to the entry with that name. The translator needs to be aware that the
 1492 incoming CBOR payload may have changed the order of the fields, when compared to the D-Bus
 1493 structure. When the version of AllJoyn implemented on the Bridged Device is less than v16.10.00,
 1494 the translator shall follow the rule for translating D-Bus structures without the aid of introspection
 1495 data.

1496 **7.2.2.8 Arrays and Dictionaries**

1497 If the introspection of the AllJoyn interface indicates that the array is neither an ARRAY of BYTE (“ay”) nor
 1498 an ARRAY of VARIANT (“av”) or that the dictionary is not mapping STRING to VARIANT (“a{sv}”), the
 1499 translator shall apply the constraining or relaxing rules specified in other sections.

1500 Similarly, if the OCF introspection indicates a homogeneous array type, the information about the array’s
 1501 element type should be used as the D-Bus array type instead of VARIANT (“v”).

1502 **7.2.2.9 Other JSON format attribute values**

1503 The JSON format attribute may include other custom attribute types. They are not known at this time, but
 1504 it is expected that those types be handled by their type and representation alone.

1505 **7.2.2.10 Examples**

AllJoyn Source	AllJoyn Introspection Notes	Translated JSON Payload	OCF Introspection Notes
UINT32 (0)		0	JSON schema should indicate: “type”: “integer”, “minimum”: 0, “maximum”: 4294967295
INT64 (0)		0	Since no Min/Max annotations exist in AllJoyn, JSON schema should indicate: “type”: “string”, “pattern”: “^0 (-?[1-9][0-9]{0,18})\$”
UINT64 (0)		“0”	Since no Max annotation exists in AllJoyn, JSON schema should indicate: “type”: “string”, “pattern”: “^0 ([1-9][0-9]{0,19})\$”
STRING(“Hello”)		“Hello”	JSON schema should indicate: “type”: “string”
OBJECT_PATH(“/”)		“/”	JSON schema should indicate: “type”: “string”
SIGNATURE(“g”)		“g”	JSON schema should indicate: “type”: “string”

AllJoyn Source	AllJoyn Introspection Notes	Translated JSON Payload	OCF Introspection Notes
ARRAY<BYTE>(0x48, 0x65, 0x6c, 0x6c, 0x6f)		"SGVsbG8"	JSON schema should indicate: "type": "string", "media binaryEncoding": "base64"
VARIANT(<i>anything</i>)		?	JSON schema should indicate: "type": ["boolean", "object", "array", "number", "string", "integer"]
ARRAY<INT32>()		[]	JSON schema should indicate: "type": "array", "items": { "type": "integer" }
ARRAY<INT64>()		[]	JSON schema should indicate: "type": "array", "items": { "type": "string", "pattern": "^0 ([1-9][0-9]{0,18})\$" }
STRUCT< INT32, INT32>(0, 1)	AllJoyn introspection specifies the argument with the annotation: <struct name="Point"> <field name="x" type="i"/> <field name="y" type="i"/> </struct>	{"x": 0, "y": 1}	JSON schema should indicate: "type": "object", "properties": { "x": { "type": "integer" }, "y": { "type": "integer" } }

1506

CBOR Payload	OCF Introspection Notes	Translated AllJoyn	AllJoyn Introspection Notes
0	"type": "integer"	INT32(0)	
0	"type": "integer", "minimum": -2 ⁴⁰ , "maximum": 2 ⁴⁰	INT64(0)	org.alljoyn.Bus.Type.Min = -2 ⁴⁰ org.alljoyn.Bus.Type.Max = 2 ⁴⁰
0	"type": "integer", "minimum": 0, "maximum": 2 ⁴⁸	UINT64(0)	org.alljoyn.Bus.Type.Max = 2 ⁴⁸
0.0	"type": "number"	DOUBLE(0.0)	

CBOR Payload	OCF Introspection Notes	Translated AllJoyn	AllJoyn Introspection Notes
[1]	JSON schema indicates: "type": "array", "items": { "type": "integer", "minimum": 0, "maximum": 2 ⁴⁶ }	ARRAY<UINT64>(1)	org.alljoyn.Bus.Type.Max = 2 ⁴⁶

1507 **8 Device Type Definitions**

1508 The required Resource Types are listed in the table below.

Device Name (informative)	Device Type (“rt”) (Normative)	Required Resource name	Required Resource Type
Bridge	oic.d.bridge	Secure Mode	oic.r.securemode
Virtual Device	oic.d.virtual	Device	oic.wk.d

1509 **9 Resource Type definitions**

1510 **9.1 List of resource types**

1511 **Table 7 Alphabetical list of resource types**

Friendly Name (informative)	Resource Type (rt)	Section
Secure Mode	oic.r.securemode	9.2
AllJoyn Object	oic.r.alljoynobject	9.3

1512

1513 **9.2 Secure Mode**

1514 **9.2.1 Introduction**

1515 This resource describes a secure mode on/off feature (on/off).

1516 A secureMode value of “true” means that the feature is on, and:

- 1517 • any Bridged Server that cannot be communicated with securely shall not have a
- 1518 corresponding Virtual OCF Server, and
- 1519 • any Bridged Client that cannot be communicated with securely shall not have a
- 1520 corresponding Virtual OCF Client.

1521 A secureMode value of “false” means that the feature is off, and:

- 1522 • any Bridged Server can have a corresponding Virtual OCF Server, and
- 1523 • any Bridged Client can have a corresponding Virtual OCF Client.

1524 **9.2.2 Example URI Path**

1525 /example/SecureModeResURI

1526 9.2.3 Resource Type

1527 The resource type (rt) is defined as: oic.r.securemode.

1528 9.2.4 RAML Definition

```
1529 #%RAML 0.8
1530 title: OCFSecureMode
1531 version: v1.0.0-20170531
1532 traits:
1533   - interface:
1534     queryParameters:
1535       if:
1536         enum: ["oic.if.rw", "oic.if.baseline"]
1537
1538 /example/SecureModeResURI:
1539   description: |
1540     This resource describes a secure mode on/off feature (on/off).
1541     A secureMode value of "true" means that the feature is on, and any Bridged Server that cannot
1542     be communicated with securely shall not have a corresponding Virtual OCF Server, and any Bridged
1543     Client that cannot be communicated with securely shall not have a corresponding Virtual OCF Client.
1544     A secureMode value of "false" means that the feature is off, any Bridged Server can have a
1545     corresponding Virtual OCF Server, and any Bridged Client can have a corresponding Virtual OCF
1546     Client.
1547
1548   is: ['interface']
1549
1550   get:
1551     description: |
1552       Retrieves the value of secureMode.
1553     responses:
1554       200:
1555         body:
1556           application/json:
1557             schema: /
1558             {
1559               "id": "https://www.openconnectivity.org/ocf-
1560 apis/bridging/schemas/oic.r.securemode.json#",
1561               "$schema": "http://json-schema.org/draft-04/schema#",
1562               "description": "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
1563 reserved.",
1564               "title": "Secure Mode",
1565               "definitions": {
1566                 "oic.r.securemode": {
1567                   "type": "object",
1568                   "properties": {
1569                     "secureMode": {
1570                       "type": "boolean",
1571                       "description": "Status of the Secure Mode"
1572                     }
1573                   }
1574                 }
1575               },
1576               "type": "object",
1577               "allOf": [
1578                 {"$ref": "../core/schemas/oic.core-schema.json#/definitions/oic.core"},
1579                 {"$ref": "#/definitions/oic.r.securemode"}
1580               ],
1581               "required": [ "secureMode" ]
1582             }
1583           example: /
```

```

1584         {
1585             "rt":          ["oic.r.securemode"],
1586             "id":          "unique_example_id",
1587             "secureMode":  false
1588         }
1589
1590     post:
1591         description: |
1592             Updates the value of secureMode.
1593
1594     body:
1595         application/json:
1596             schema: /
1597                 {
1598                     "id": "https://www.openconnectivity.org/ocf-
apis/bridging/schemas/oic.r.securemode.json#",
1599                     "$schema": "http://json-schema.org/draft-04/schema#",
1600                     "description" : "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
1601 reserved.",
1602                     "title": "Secure Mode",
1603                     "definitions": {
1604                         "oic.r.securemode": {
1605                             "type": "object",
1606                             "properties": {
1607                                 "secureMode": {
1608                                     "type": "boolean",
1609                                     "description": "Status of the Secure Mode"
1610                                 }
1611                             }
1612                         },
1613                     },
1614                     "type": "object",
1615                     "allOf": [
1616                         { "$ref": "../core/schemas/oic.core-schema.json#/definitions/oic.core" },
1617                         { "$ref": "#/definitions/oic.r.securemode" }
1618                     ],
1619                     "required": [ "secureMode" ]
1620                 }
1621
1622     example: /
1623         {
1624             "id":          "unique_example_id",
1625             "secureMode":  true
1626         }
1627
1628     responses:
1629         200:
1630             body:
1631                 application/json:
1632                     schema: /
1633                         {
1634                             "id": "https://www.openconnectivity.org/ocf-
apis/bridging/schemas/oic.r.securemode.json#",
1635                             "$schema": "http://json-schema.org/draft-04/schema#",
1636                             "description" : "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
1637 reserved.",
1638                             "title": "Secure Mode",
1639                             "definitions": {
1640                                 "oic.r.securemode": {
1641                                     "type": "object",
1642                                     "properties": {
1643                                         "secureMode": {
1644                                             "type": "boolean",
1645                                             "description": "Status of the Secure Mode"
1646                                         }
1647                                     }
1648                                 }
1649                             }
1650                         }

```

```

1648     },
1649     "type": "object",
1650     "allOf": [
1651         {"$ref": "../core/schemas/oic.core-schema.json#/definitions/oic.core"},
1652         {"$ref": "#/definitions/oic.r.securemode"}
1653     ],
1654     "required": [ "secureMode" ]
1655 }
1656
1657 example: /
1658 {
1659     "id": "unique_example_id",
1660     "secureMode": true
1661 }
1662

```

9.2.5 Swagger2.0 Definition

```

1664 {
1665     "swagger": "2.0",
1666     "info": {
1667         "title": "OCFSecureMode",
1668         "version": "v1.0.0-20170531",
1669         "license": {
1670             "name": "copyright 2016-2017 Open Connectivity Foundation, Inc. All rights reserved.",
1671             "x-description": "Redistribution and use in source and binary forms, with or without
1672 modification, are permitted provided that the following conditions are met:\n    1.
1673 Redistributions of source code must retain the above copyright notice, this list of conditions and
1674 the following disclaimer.\n    2. Redistributions in binary form must reproduce the above
1675 copyright notice, this list of conditions and the following disclaimer in the documentation and/or
1676 other materials provided with the distribution.\n\n    THIS SOFTWARE IS PROVIDED BY THE Open
1677 Connectivity Foundation, INC. \AS IS\ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
1678 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR
1679 WARRANTIES OF NON-INFRINGEMENT, ARE DISCLAIMED.\n    IN NO EVENT SHALL THE Open Connectivity
1680 Foundation, INC. OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
1681 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
1682 OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)\n    HOWEVER CAUSED AND
1683 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
1684 OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
1685 OF SUCH DAMAGE.\n"
1686         }
1687     },
1688     "schemes": ["http"],
1689     "consumes": ["application/json"],
1690     "produces": ["application/json"],
1691     "paths": {
1692         "/example/SecureModeResURI" : {
1693             "get": {
1694                 "description": "This resource describes a secure mode on/off feature (on/off).\nA
1695 secureMode value of 'true' means that the feature is on, and any Bridged Server that cannot be
1696 communicated with securely shall not have a corresponding Virtual OCF Server, and any Bridged
1697 Client that cannot be communicated with securely shall not have a corresponding Virtual OCF
1698 Client.\nA secureMode value of 'false' means that the feature is off, any Bridged Server can have a
1699 corresponding Virtual OCF Server, and any Bridged Client can have a corresponding Virtual OCF
1700 Client.\nRetrieves the value of secureMode.\n",
1701                 "parameters": [
1702                     {"$ref": "#/parameters/interface"}
1703                 ],
1704                 "responses": {
1705                     "200": {
1706                         "description": "",
1707                         "x-example": {
1708                             "rt": ["oic.r.securemode"],
1709                             "id": "unique_example_id",
1710                             "secureMode": false
1711                         }
1712                     },
1713                     "": {}
1714                 },
1715                 "schema": {"$ref": "#/definitions/SecureMode"}
1716             }
1717         }
1718     }
1719 }

```

```

1717     },
1718     "post": {
1719         "description": "Updates the value of secureMode.\n",
1720         "parameters": [
1721             { "$ref": "#/parameters/interface" },
1722             {
1723                 "name": "body",
1724                 "in": "body",
1725                 "required": true,
1726                 "schema": { "$ref": "#/definitions/SecureMode" },
1727                 "x-example":
1728                     {
1729                         "id": "unique_example_id",
1730                         "secureMode": true
1731                     }
1732             }
1733         ],
1734         "responses": {
1735             "200": {
1736                 "description": "",
1737                 "x-example":
1738                     {
1739                         "id": "unique_example_id",
1740                         "secureMode": true
1741                     }
1742             },
1743             "schema": { "$ref": "#/definitions/SecureMode" }
1744         }
1745     }
1746 },
1747 },
1748 },
1749 "parameters": {
1750     "interface": {
1751         "in": "query",
1752         "name": "if",
1753         "type": "string",
1754         "enum": ["oic.if.rw", "oic.if.baseline"]
1755     }
1756 },
1757 "definitions": {
1758     "SecureMode": {
1759         {
1760             "properties": {
1761                 "id": {
1762                     "description": "Instance ID of this specific resource",
1763                     "maxLength": 64,
1764                     "readOnly": true,
1765                     "type": "string"
1766                 },
1767                 "if": {
1768                     "description": "The interface set supported by this resource",
1769                     "items": {
1770                         "enum": [
1771                             "oic.if.baseline",
1772                             "oic.if.ll",
1773                             "oic.if.b",
1774                             "oic.if.lb",
1775                             "oic.if.rw",
1776                             "oic.if.r",
1777                             "oic.if.a",
1778                             "oic.if.s"
1779                         ],
1780                         "type": "string"
1781                     },
1782                     "minItems": 1,
1783                     "readOnly": true,
1784                     "type": "array"
1785                 }
1786             },
1787             "n": {
1788                 "description": "Friendly name of the resource",

```

```

1788     "maxLength": 64,
1789     "readOnly": true,
1790     "type": "string"
1791   },
1792   "rt": {
1793     "description": "Resource Type",
1794     "items": {
1795       "maxLength": 64,
1796       "type": "string"
1797     },
1798     "minItems": 1,
1799     "readOnly": true,
1800     "type": "array"
1801   },
1802   "secureMode": {
1803     "description": "Status of the Secure Mode",
1804     "type": "boolean"
1805   }
1806 },
1807 "required": [
1808   "secureMode"
1809 ],
1810 "type": "object"
1811 }
1812 }
1813 }
1814 }
1815 }
1816 }

```

1817 9.2.6 Property Definition

Property name	Value type	Mandatory	Access mode	Description
secureMode	boolean	Yes	Read Write	Status of the Secure Mode

1818 9.2.7 CRUDN behaviour

Example Resource URI	Create	Read	Update	Delete	Notify
/example/SecureModeResURI		get	post		get

1819

1820 9.3 AllJoyn Object

1821 9.3.1 Introduction

1822 This resource is a collection of resources that were all derived from the same AllJoyn object.

1823 9.3.2 Example URI Path

1824 /example/AllJoynObject/

1825 9.3.3 Resource Type

1826 The resource type (rt) is defined as: oic.r.alljoynobject.

1827 9.3.4 RAML Definition

```

1828 #%RAML 0.8
1829 title: OCFAllJoynObject
1830 version: v1.0.0-20170531
1831 traits:
1832 - interface-11:
1833   queryParameters:
1834     if:
1835       enum: ["oic.if.11"]
1836 - interface-baseline:
1837   queryParameters:

```

```

1838     if:
1839         enum: ["oic.if.baseline"]
1840 - interface-all:
1841     queryParameters:
1842         if:
1843             enum: ["oic.if.ll", "oic.if.baseline"]
1844
1845 /example/AllJoynObject/?if=oic.if.baseline:
1846     description: |
1847         This resource is a collection of resources that were all derived from the same AllJoyn object.
1848
1849     is: ['interface-baseline']
1850
1851     get:
1852         description: |
1853             Retrieves the current AllJoyn object information.
1854     responses:
1855         200:
1856             body:
1857                 application/json:
1858                     schema: /
1859                         {
1860                             "id": "https://www.openconnectivity.org/ocf-
1861 apis/bridging/schemas/oic.r.alljoynobject.json#",
1862                             "$schema": "http://json-schema.org/draft-04/schema#",
1863                             "description": "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
1864 reserved.",
1865                             "title": "AllJoyn Object",
1866                             "definitions": {
1867                                 "oic.r.alljoynobject": {
1868                                     "type": "object",
1869                                     "allOf": [
1870                                         {
1871                                             "$ref": "../../core/schemas/oic.collection-
1872 schema.json#/definitions/oic.collection"
1873                                         },
1874                                         {
1875                                             "properties": {
1876                                                 "rt": {
1877                                                     "type": "array",
1878                                                     "minItems": 2,
1879                                                     "maxItems": 2,
1880                                                     "uniqueItems": true,
1881                                                     "items": {
1882                                                         "enum": ["oic.r.alljoynobject", "oic.wk.col"]
1883                                                     }
1884                                                 }
1885                                             }
1886                                         }
1887                                     ]
1888                                 }
1889                             },
1890                             "type": "object",
1891                             "allOf": [
1892                                 {"$ref": "../../core/schemas/oic.core-schema.json#/definitions/oic.core"},
1893                                 {"$ref": "#/definitions/oic.r.alljoynobject"}
1894                             ]
1895                         }
1896     example: /
1897         {
1898             "rt": ["oic.r.alljoynobject", "oic.wk.col"],
1899             "id": "unique_example_id",

```

```

1900         "links": [
1901             { "href": "/myRes1URI", "rt": ["x.example.widget.false"], "if":
1902 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1903             { "href": "/myRes2URI", "rt": ["x.example.widget.true"], "if":
1904 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1905             { "href": "/myRes3URI", "rt": ["x.example.widget.method1"], "if":
1906 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1907             { "href": "/myRes4URI", "rt": ["x.example.widget.method2"], "if":
1908 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]}
1909         ]
1910     }
1911 /example/AllJoynObject/?if=oic.if.ll:
1912     description: |
1913     This resource is a collection of resources that were all derived from the same AllJoyn object.
1914
1915     is: ['interface-ll']
1916
1917     get:
1918     description: |
1919     Retrieves the Links in the current AllJoyn object.
1920
1921     responses:
1922     200:
1923     body:
1924     application/json:
1925         schema: /
1926         {
1927             "id": "https://www.openconnectivity.org/ocf-
apis/bridging/schemas/oic.r.alljoynobject-ll#",
1928             "$schema": "http://json-schema.org/draft-04/schema#",
1929             "description": "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
reserved.",
1930             "title": "AllJoyn Object Links List Schema",
1931             "definitions": {
1932                 "oic.r.alljoynobject-ll": {
1933                     "allOf": [
1934                         {
1935                             "$ref": "../../core/schemas/oic.collection.linkslist-
schema.json#/definitions/oic.collection.alllinks"
1936                         }
1937                     ]
1938                 }
1939             }
1940         },
1941         "allOf": [
1942             {"$ref": "#/definitions/oic.r.alljoynobject-ll"}
1943         ]
1944     }
1945
1946     example: /
1947     [
1948         { "href": "/myRes1URI", "rt": ["x.example.widget.false"], "if":
1949 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1950         { "href": "/myRes2URI", "rt": ["x.example.widget.true"], "if":
1951 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1952         { "href": "/myRes3URI", "rt": ["x.example.widget.method1"], "if":
1953 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1954         { "href": "/myRes4URI", "rt": ["x.example.widget.method2"], "if":
1955 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]}
1956     ]
1957
1958

```

1959 9.3.5 Swagger2.0 Definition

```

1960 {
1961     "swagger": "2.0",
1962     "info": {

```

```

1963     "title": "OCFAllJoynObject",
1964     "version": "v1.0.0-20170531",
1965     "license": {
1966         "name": "copyright 2016-2017 Open Connectivity Foundation, Inc. All rights reserved.",
1967         "x-description": "Redistribution and use in source and binary forms, with or without
1968 modification, are permitted provided that the following conditions are met:\n      1.
1969 Redistributions of source code must retain the above copyright notice, this list of conditions and
1970 the following disclaimer.\n      2. Redistributions in binary form must reproduce the above
1971 copyright notice, this list of conditions and the following disclaimer in the documentation and/or
1972 other materials provided with the distribution.\n\n      THIS SOFTWARE IS PROVIDED BY THE Open
1973 Connectivity Foundation, INC. \AS IS\ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
1974 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR
1975 WARRANTIES OF NON-INFRINGEMENT, ARE DISCLAIMED.\n      IN NO EVENT SHALL THE Open Connectivity
1976 Foundation, INC. OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
1977 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
1978 OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)\n      HOWEVER CAUSED AND
1979 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
1980 OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
1981 OF SUCH DAMAGE.\n"
1982     }
1983 },
1984 "schemes": ["http"],
1985 "consumes": ["application/json"],
1986 "produces": ["application/json"],
1987 "paths": {
1988     "/example/AllJoynObject/?if=oic.if.ll" : {
1989         "get": {
1990             "description": "This resource is a collection of resources that were all derived from the
1991 same AllJoyn object.\nRetrieves the Links in the current AllJoyn object.\n",
1992             "parameters": [
1993                 {"$ref": "#/parameters/interface-ll"}
1994             ],
1995             "responses": {
1996                 "200": {
1997                     "description": "",
1998                     "x-example":
1999                     [
2000                         {"href": "/myRes1URI", "rt": ["x.example.widget.false"], "if":
2001 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
2002                         {"href": "/myRes2URI", "rt": ["x.example.widget.true"], "if":
2003 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
2004                         {"href": "/myRes3URI", "rt": ["x.example.widget.method1"], "if":
2005 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
2006                         {"href": "/myRes4URI", "rt": ["x.example.widget.method2"], "if":
2007 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]}
2008                     ]
2009                 },
2010                 "schema": {"$ref": "#/definitions/AllJoynObject-ll" }
2011             }
2012         }
2013     }
2014 },
2015     "/example/AllJoynObject/?if=oic.if.baseline" : {
2016         "get": {
2017             "description": "This resource is a collection of resources that were all derived from the
2018 same AllJoyn object.\nRetrieves the current AllJoyn object information.\n",
2019             "parameters": [
2020                 {"$ref": "#/parameters/interface-baseline"}
2021             ],
2022             "responses": {
2023                 "200": {
2024                     "description": "",
2025                     "x-example":
2026                     {
2027                         "rt": ["oic.r.alljoynobject", "oic.wk.col"],
2028                         "id": "unique_example_id",
2029                         "links": [
2030                             {"href": "/myRes1URI", "rt": ["x.example.widget.false"], "if":
2031 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
2032                             {"href": "/myRes2URI", "rt": ["x.example.widget.true"], "if":
2033 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},

```

```

2034         {"href": "/myRes3URI", "rt": ["x.example.widget.method1"], "if":
2035 ["oic.if.rw","oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a:b1d4]:11111"}]},
2036         {"href": "/myRes4URI", "rt": ["x.example.widget.method2"], "if":
2037 ["oic.if.rw","oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a:b1d4]:11111"}]}
2038     ]
2039 }
2040 ,
2041     "schema": { "$ref": "#/definitions/AllJoynObject" }
2042 }
2043 }
2044 }
2045 }
2046 },
2047 "parameters": {
2048     "interface-ll" : {
2049         "in" : "query",
2050         "name" : "if",
2051         "type" : "string",
2052         "enum" : ["oic.if.ll"]
2053     },
2054     "interface-baseline" : {
2055         "in" : "query",
2056         "name" : "if",
2057         "type" : "string",
2058         "enum" : ["oic.if.baseline"]
2059     },
2060     "interface-all" : {
2061         "in" : "query",
2062         "name" : "if",
2063         "type" : "string",
2064         "enum" : ["oic.if.ll", "oic.if.baseline"]
2065     }
2066 },
2067 "definitions": {
2068     "AllJoynObject-ll" :
2069     {
2070         "allOf": [
2071         {
2072             "$ref": "../../core/schemas/oic.collection.linkslist-schema.json"
2073         }
2074         ]
2075     }
2076 },
2077 ,
2078 "AllJoynObject" :
2079 {
2080     "allOf": [
2081     {
2082         "$ref": "../../core/schemas/oic.collection-schema.json"
2083     },
2084     {
2085         "properties": {
2086             "id": {
2087                 "description": "Instance ID of this specific resource",
2088                 "maxLength": 64,
2089                 "readOnly": true,
2090                 "type": "string"
2091             },
2092             "if": {
2093                 "description": "The interface set supported by this resource",
2094                 "items": {
2095                     "enum": [
2096                         "oic.if.baseline",
2097                         "oic.if.ll",
2098                         "oic.if.b",
2099                         "oic.if.lb",
2100                         "oic.if.rw",
2101                         "oic.if.r",
2102                         "oic.if.a",
2103                         "oic.if.s"
2104                     ]
2105                 }
2106             }
2107         }
2108     }
2109 }

```

```

2105         "type": "string"
2106     },
2107     "minItems": 1,
2108     "readOnly": true,
2109     "type": "array"
2110 },
2111 "n": {
2112     "description": "Friendly name of the resource",
2113     "maxLength": 64,
2114     "readOnly": true,
2115     "type": "string"
2116 },
2117 "rt": {
2118     "items": {
2119         "enum": [
2120             "oic.r.alljoynobject",
2121             "oic.wk.col"
2122         ]
2123     },
2124     "maxItems": 2,
2125     "minItems": 2,
2126     "type": "array",
2127     "uniqueItems": true
2128 }
2129 }
2130 }
2131 ],
2132 "type": "object"
2133 }
2134 }
2135 }
2136 }
2137 }

```

2138 **9.3.6 CRUDN behaviour**

Example Resource URI	Create	Read	Update	Delete	Notify
/example/AllJoynObject/?if=oic.if.baseline		get	post		get
/example/AllJoynObject/?if=oic.if.ll		get			get

2139