

OCF Bridging Specification

VERSION 1.0.0 | June 2017



OPEN CONNECTIVITY
FOUNDATION™

CONTACT admin@openconnectivity.org

Copyright Open Connectivity Foundation, Inc. © 2017.
All Rights Reserved.

4 Legal Disclaimer

5

6 NOTHING CONTAINED IN THIS DOCUMENT SHALL BE DEEMED AS GRANTING YOU ANY
7 KIND OF LICENSE IN ITS CONTENT, EITHER EXPRESSLY OR IMPLIEDLY, OR TO ANY
8 INTELLECTUAL PROPERTY OWNED OR CONTROLLED BY ANY OF THE AUTHORS OR
9 DEVELOPERS OF THIS DOCUMENT. THE INFORMATION CONTAINED HEREIN IS PROVIDED
10 ON AN "AS IS" BASIS, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW,
11 THE AUTHORS AND DEVELOPERS OF THIS SPECIFICATION HEREBY DISCLAIM ALL OTHER
12 WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, STATUTORY OR AT
13 COMMON LAW, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
14 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. OPEN CONNECTIVITY
15 FOUNDATION, INC. FURTHER DISCLAIMS ANY AND ALL WARRANTIES OF NON-
16 INFRINGEMENT, ACCURACY OR LACK OF VIRUSES.

17

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

20

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

22

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

24

25
26
27
28
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

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
4	Document conventions and organization	9
4.1	Conventions	9
4.2	Notation.....	9
4.3	Data types	10
4.4	Document structure	10
5	Operational Scenarios.....	10
5.1	Goals.....	10
5.2	“Deep translation” vs. “on-the-fly”	10
5.3	Use of introspection.....	11
5.4	Stability and loss of data	11
6	OCF Bridge Device	11
6.1	Introduction	11
6.2	Resource Discovery.....	13
6.3	General Requirements.....	22
6.4	Security.....	22
6.4.1	General Security Considerations	22
6.4.2	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	Introduction	23
7.1.2	Exposing AllJoyn producer devices to OCF Clients	23
7.1.3	Exposing OCF resources to AllJoyn consumer applications.....	31
7.2	On-the-Fly Translation from D-Bus and OCF payloads.....	36
7.2.1	Introduction	36
7.2.2	Translation without aid of introspection	36
7.2.3	Translation with aid of introspection	42
8	Device Type Definitions.....	47
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	48
9.2.4	RAML Definition	48

67	9.2.5	Swagger2.0 Definition	50
68	9.2.6	Property Definition	53
69	9.2.7	CRUDN behaviour	53
70	9.3	AllJoyn Object	53
71	9.3.1	Introduction	53
72	9.3.2	Example URI Path	53
73	9.3.3	Resource Type	53
74	9.3.4	RAML Definition	53
75	9.3.5	Swagger2.0 Definition	55
76	9.3.6	CRUDN behaviour	58
77			

Figures

78		
79	Figure 1. OCF Bridge Device Components	12
80	Figure 2: Schematic overview of an OCF Bridge Device bridging non-OCF devices	12
81		

Tables

82		
83	Table 7 Alphabetical list of resource types.....	48
84		
85	Table 1: oic.wk.d resource type definition	26
86	Table 2: oic.wk.con resource type definition	28
87	Table 3: oic.wk.p Resource Type definition.....	29
88	Table 4: oic.wk.con.p Resource Type definition	30
89	Table 5: AllJoyn About Data fields	33
90	Table 6: AllJoyn Configuration Data fields	35
91	Table 7 Alphabetical list of resource types.....	48
92		
93		

94 **1 Scope**

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

101 **2 Normative references**

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

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

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

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

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

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

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

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

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

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

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

124 JSON Schema Validation, *JSON Schema: interactive and non interactive validation*, January
125 2013
126 <http://json-schema.org/latest/json-schema-validation.html>

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

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

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

132 OCF ASA Mapping, *OCF Resource to ASA Interface Mapping*, v0.3 candidate, July 2016
133 [https://workspace.openconnectivity.org/apps/org/workgroup/smarthome_tg/download.php/6287/O](https://workspace.openconnectivity.org/apps/org/workgroup/smarthome_tg/download.php/6287/OCF_Resource_to_ASA_Interface_Mapping_v.0.3_candidate.docx)
134 [CF_Resource_to_ASA_Interface_Mapping_v.0.3_candidate.docx](https://workspace.openconnectivity.org/apps/org/workgroup/smarthome_tg/download.php/6287/OCF_Resource_to_ASA_Interface_Mapping_v.0.3_candidate.docx)

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

136 RAML Specification, *Restful API modelling language*, Version 0.8.
137 <https://github.com/raml-org/raml-spec/blob/master/versions/raml-08/raml-08.md>

138 OCF Resource Type Definitions, *API Definition Language for OCF Resource Type Definitions*,
139 Release OCF-v1.0.0
140 <https://github.com/openconnectivityfoundation/bridging>

141

142 **3 Terms, definitions, symbols and abbreviations**

143 **3.1 Terms and definitions**

144 **3.1.1**

145 **OCF Bridge Device**

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

148 **3.1.2**

149 **Bridged Protocol**

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

151 **3.1.3**

152 **Translator**

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

156 **3.1.4**

157 **OCF Client**

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

160 **3.1.5**

161 **Bridged Client**

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

164 **3.1.6**

165 **Virtual OCF Client**

166 a logical representation of a Bridged Client, which an OCF Bridge Device exposes to OCF Servers.

167 **3.1.7**

168 **Virtual Bridged Client**

169 a logical representation of an OCF Client, which an OCF Bridge Device exposes to Bridged Servers.

170 **3.1.8**

171 **OCF Device**

172 a logical entity that assumes one or more OCF roles (OCF Client, OCF Server). More than one
173 OCF Device can exist on the same physical platform.

174 **3.1.9**
175 **Virtual OCF Server**
176 a logical representation of a Bridged Server, which an OCF Bridge Device exposes to OCF Clients.

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

181 **3.1.11**
182 **Virtual Bridged Server**
183 a logical representation of an OCF Server, which an OCF Bridge Device exposes to Bridged Clients.

184 **3.1.12**
185 **OCF Resource**
186 represents an artifact modelled and exposed by the OCF Framework

187 **3.1.13**
188 **Virtual OCF Resource**
189 a logical representation of a Bridged Resource, which an OCF Bridge Device exposes to OCF
190 Clients.

191 **3.1.14**
192 **Bridged Resource**
193 represents an artifact modelled and exposed by a Bridged Protocol. For example, an AllJoyn
194 object is a Bridged Resource.

195 **3.1.15**
196 **OCF Resource Property**
197 a significant aspect or notion including metadata that is exposed through the OCF Resource

198 **3.1.16**
199 **OCF Resource Type**
200 an OCF Resource Property that represents the data type definition for the OCF Resource

201 **3.1.17**
202 **Bridged Resource Type**
203 a schema used with a Bridged Protocol. For example, AllJoyn Interfaces are Bridged Resource
204 Types.

205 **3.1.18**
206 **OCF Server**
207 a logical entity with the role of providing resource state information and allowing remote control of
208 its resources.

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

214 **3.1.20**
215 **Bridged Device**
216 a Bridged Client or Bridged Server.

217 **3.1.21**
218 **Virtual OCF Device**
219 a Virtual OCF Client or Virtual OCF Server.

220 **3.1.22**
221 **CRUDN**
222 Create Read Update Delete Notify
223 indicating which operations are possible on the resource

224 **3.1.23**
225 **CSV**
226 Comma Separated Value List
227 construction to have more fields in 1 string separated by commas. If a value contains a comma,
228 then the comma can be escaped by adding “\” in front of the comma.

229 **3.1.24**
230 **OCF**
231 Open Connectivity Foundation
232 organization that created these specifications

233 **3.1.25**
234 **RAML**
235 RESTful API Modeling Language
236 Simple and succinct way of describing practically RESTful APIs (see the OIC 1.1 Core
237 Specification, *Open Interconnect Consortium Core Specification, Version 1.1*

238 **3.2 RAML Specification)Symbols and abbreviations**
239 None defined.

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 Conventions**

244 **4.2 In this specification several terms, conditions, mechanisms, sequences, parameters,**
245 **events, states, or similar terms are printed with the first letter of each word in**
246 **uppercase and the rest lowercase (e.g., Network Architecture). Any lowercase uses**
247 **of these words have the normal technical English meaning.**Notation

248 In this document, features are described as required, recommended, allowed or DEPRECATED as
249 follows:

250 Required (or shall or mandatory).

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

254 Recommended (or should).

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

261 Allowed (or allowed).

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

264 Conditionally allowed (CA)

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

267 Conditionally required (CR)

268 – The definition or behaviour depends on a condition. If the specified condition is met, then the
269 definition or behaviour is required. Otherwise the definition or behaviour is allowed as default
270 unless specifically defined as not allowed.

271 DEPRECATED

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

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

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

280 **4.3 Data types**

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

282 **4.4 Document structure**

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

287 **5 Operational Scenarios**

288 **5.1 Goals**

289 The overall goals are to:

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

292 **5.2 “Deep translation” vs. “on-the-fly”**

293 When translating a service between a Bridged Protocol (e.g., AllJoyn) and OCF protocols, there
294 are two possible types of translation. Translators are expected to dedicate most of their logic to
295 “deep translation” types of communication, in which data models used with the Bridged Protocol
296 are mapped to the equivalent OCF Resource Types and vice-versa, in such a way that a compliant
297 OCF Client or Bridged Client would be able to interact with the service without realising that a
298 translation was made.

299 “Deep translation” is out of the scope of this document, as the procedure far exceeds mapping of
300 types. For example, clients on one side of a translator may decide to represent an intensity as an
301 8-bit value between 0 and 255, whereas the devices on the other may have chosen to represent
302 that as a floating-point number between 0.0 and 1.0. It’s also possible that the procedure may

303 require storing state in the translator. Either way, the programming of such translation will require
304 dedicated effort and study of the mechanisms on both sides.

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

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

312 **5.3 Use of introspection**

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

320 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

321 **5.4 Stability and loss of data**

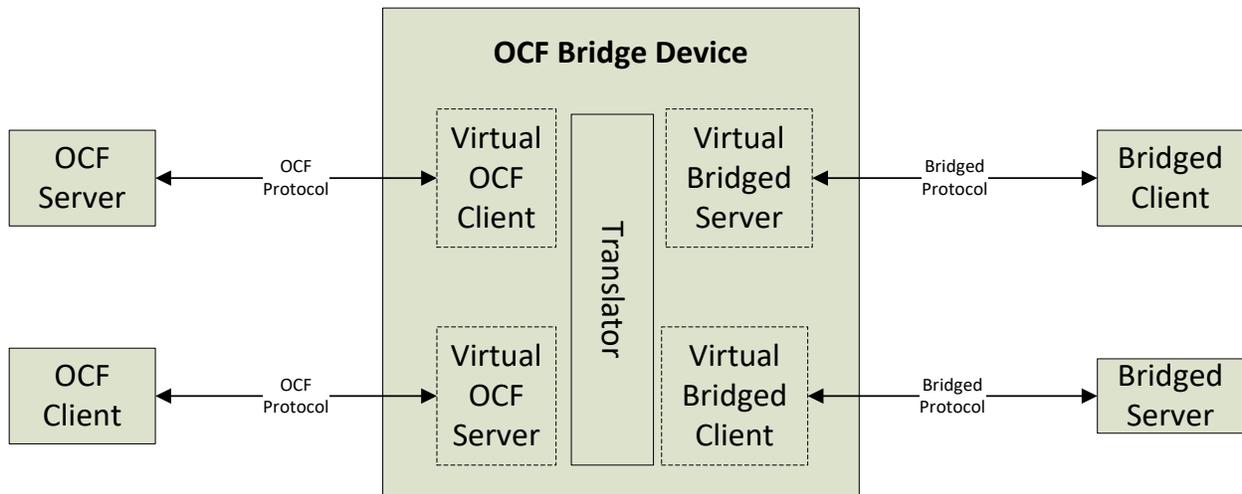
322 Round-tripping through the translation process specified in this document is not expected to
323 reproduce the same original message. The process is, however, designed not to lose data or
324 precision in messages, though it should be noted that both OCF and AllJoyn payload formats allow
325 for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

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

329 **6 OCF Bridge Device**

330 **6.1 Introduction**

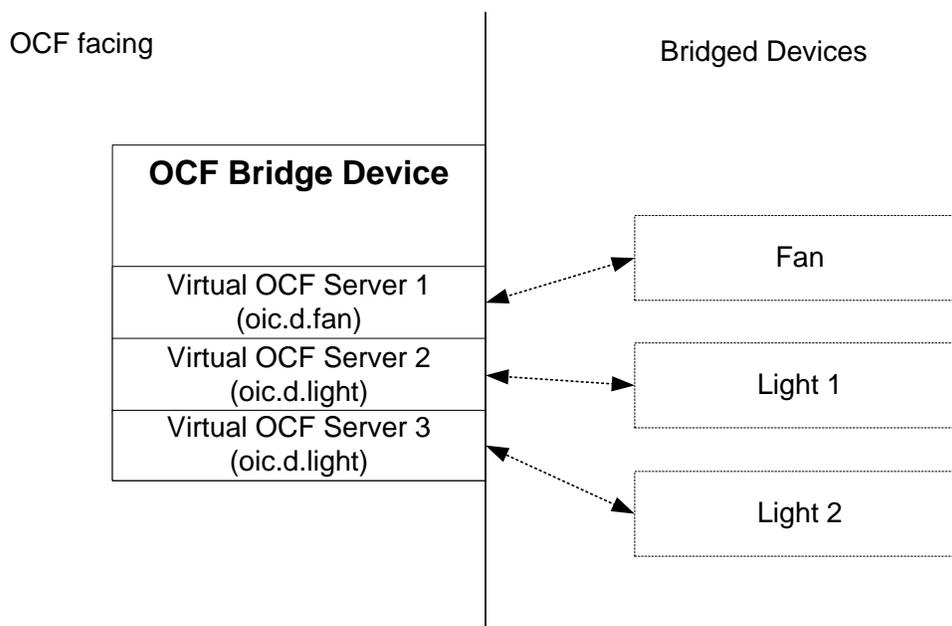
331 This section describes the functionality of an OCF Bridge Device; such a device is illustrated in
332 Figure 1. OCF Bridge Device Components.



333

334 Figure 1. OCF Bridge Device Components An OCF Bridge Device is a device that represents one
 335 or more Bridged Devices as Virtual OCF Devices on the network and/or represents one or more
 336 OCF Devices as Virtual Devices using another protocol on the network. The Bridged Devices
 337 themselves are out of the scope of this document. The only difference between a native OCF
 338 Device and a Virtual Bridged Device is how the device is encapsulated in an OCF Bridge Device.

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



348

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

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

355 **6.2 Resource Discovery**

356 An OCF Bridge Device shall detect devices that arrive and leave the Bridged network or the OCF
357 network. Where there is no pre-existing mechanism to reliably detect the arrival and departure of
358 devices on a network, an OCF Bridge Device shall periodically poll the network to detect arrival
359 and departure of devices, for example using COAP multicast discovery (a multicast RETRIEVE of
360 "/oic/res") in the case of the OCF network. OCF Bridge Device implementations are encouraged to
361 use a poll interval of 30 seconds plus or minus a random delay of a few seconds.

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

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

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

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

```

405     "rt": ["oic.r.doxm"],
406     "if": ["oic.if.baseline"],
407     "p": {"bm": 1, "sec": true, "port": 11111}
408   },
409   {
410     "href": "/oic/sec/pstat",
411     "rt": ["oic.r.pstat"],
412     "if": ["oic.if.baseline"],
413     "p": {"bm": 1, "sec": true, "port": 11111}
414   },
415   {
416     "href": "/oic/sec/cred",
417     "rt": ["oic.r.cred"],
418     "if": ["oic.if.baseline"],
419     "p": {"bm": 1, "sec": true, "port": 11111}
420   },
421   {
422     "href": "/oic/sec/acl2",
423     "rt": ["oic.r.acl2"],
424     "if": ["oic.if.baseline"],
425     "p": {"bm": 1, "sec": true, "port": 11111}
426   },
427   {
428     "href": "/myIntrospection",
429     "rt": ["oic.wk.introspection"],
430     "if": ["oic.if.r", "oic.if.baseline"],
431     "p": {"bm": 3, "sec": true, "port": 11111}
432   }
433 ]
434 },
435 {
436   "di": "88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
437   "links": [
438     {
439       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/res",
440       "rt": ["oic.wk.res"],
441       "if": ["oic.if.ll", "oic.if.baseline"],
442       "p": {"bm": 3, "sec": true, "port": 22222}
443     },
444     {
445       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/d",
446       "rt": ["oic.wk.d", "oic.d.fan", "oic.d.virtual"],
447       "if": ["oic.if.r", "oic.if.baseline"],
448       "p": {"bm": 3, "sec": true, "port": 22222}
449     },
450     {
451       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/p",
452       "rt": ["oic.wk.p"],
453       "if": ["oic.if.r", "oic.if.baseline"],
454       "p": {"bm": 3, "sec": true, "port": 22222}
455     },
456     {
457       "href": "coaps://[2001:db8:a::b1d4]:22222/myFan",
458       "rt": ["oic.r.switch.binary"],
459       "if": ["oic.if.a", "oic.if.baseline"],
460       "p": {"bm": 3, "sec": true, "port": 22222}
461     },
462     {
463       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/sec/doxm",
464       "rt": ["oic.r.doxm"],
465       "if": ["oic.if.baseline"],
466       "p": {"bm": 1, "sec": true, "port": 22222}
467     },

```

```

468     {
469       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/sec/pstat",
470       "rt": ["oic.r.pstat"],
471       "if": ["oic.if.baseline"],
472       "p": {"bm": 1, "sec": true, "port": 22222}
473     },
474     {
475       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/sec/cred",
476       "rt": ["oic.r.cred"],
477       "if": ["oic.if.baseline"],
478       "p": {"bm": 1, "sec": true, "port": 22222}
479     },
480     {
481       "href": "coaps://[2001:db8:a::b1d4]:22222/oic/sec/acl2",
482       "rt": ["oic.r.acl2"],
483       "if": ["oic.if.baseline"],
484       "p": {"bm": 1, "sec": true, "port": 22222}
485     },
486     {
487       "href": "coaps://[2001:db8:a::b1d4]:22222/myFanIntrospection",
488       "rt": ["oic.wk.introspection"],
489       "if": ["oic.if.r", "oic.if.baseline"],
490       "p": {"bm": 3, "sec": true, "port": 22222}
491     }
492   ]
493 },
494 {
495   "di": "dc70373c-1e8d-4fb3-962e-017eaa863989",
496   "links": [
497     {
498       "href": "coaps://[2001:db8:a::b1d4]:33333/oic/res",
499       "rt": ["oic.wk.res"],
500       "if": ["oic.if.ll", "oic.if.baseline"],
501       "p": {"bm": 3, "sec": true, "port": 33333}
502     },
503     {
504       "href": "coaps://[2001:db8:a::b1d4]:33333/oic/d",
505       "rt": ["oic.wk.d", "oic.d.light", "oic.d.virtual"],
506       "if": ["oic.if.r", "oic.if.baseline"],
507       "p": {"bm": 3, "sec": true, "port": 33333}
508     },
509     {
510       "href": "coaps://[2001:db8:a::b1d4]:33333/oic/p",
511       "rt": ["oic.wk.p"],
512       "if": ["oic.if.r", "oic.if.baseline"],
513       "p": {"bm": 3, "sec": true, "port": 33333}
514     },
515     {
516       "href": "coaps://[2001:db8:a::b1d4]:33333/myLight",
517       "rt": ["oic.r.switch.binary"],
518       "if": ["oic.if.a", "oic.if.baseline"],
519       "p": {"bm": 3, "sec": true, "port": 33333}
520     },
521     {
522       "href": "coaps://[2001:db8:a::b1d4]:33333/oic/sec/doxm",
523       "rt": ["oic.r.doxm"],
524       "if": ["oic.if.baseline"],
525       "p": {"bm": 1, "sec": true, "port": 33333}
526     },
527     {
528       "href": "coaps://[2001:db8:a::b1d4]:33333/oic/sec/pstat",
529       "rt": ["oic.r.pstat"],
530       "if": ["oic.if.baseline"],

```

```

531     "p": {"bm": 1, "sec": true, "port": 33333}
532   },
533   {
534     "href": "coaps://[2001:db8:a::b1d4]:33333/oic/sec/cred",
535     "rt": ["oic.r.cred"],
536     "if": ["oic.if.baseline"],
537     "p": {"bm": 1, "sec": true, "port": 33333}
538   },
539   {
540     "href": "coaps://[2001:db8:a::b1d4]:33333/oic/sec/acl2",
541     "rt": ["oic.r.acl2"],
542     "if": ["oic.if.baseline"],
543     "p": {"bm": 1, "sec": true, "port": 33333}
544   },
545   {
546     "href": "coaps://[2001:db8:a::b1d4]:33333/myLightIntrospection",
547     "rt": ["oic.wk.introspection"],
548     "if": ["oic.if.r", "oic.if.baseline"],
549     "p": {"bm": 3, "sec": true, "port": 33333}
550   }
551 ]
552 },
553 {
554   "di": "8202138e-aa22-452c-b512-9ebad02bef7c",
555   "links": [
556     {
557       "href": "coaps://[2001:db8:a::b1d4]:44444/oic/res",
558       "rt": ["oic.wk.res"],
559       "if": ["oic.if.ll", "oic.if.baseline"],
560       "p": {"bm": 3, "sec": true, "port": 44444}
561     },
562     {
563       "href": "coaps://[2001:db8:a::b1d4]:44444/oic/d",
564       "rt": ["oic.wk.d", "oic.d.light", "oic.d.virtual"],
565       "if": ["oic.if.r", "oic.if.baseline"],
566       "p": {"bm": 3, "sec": true, "port": 44444}
567     },
568     {
569       "href": "coaps://[2001:db8:a::b1d4]:44444/oic/p",
570       "rt": ["oic.wk.p"],
571       "if": ["oic.if.r", "oic.if.baseline"],
572       "p": {"bm": 3, "sec": true, "port": 44444}
573     },
574     {
575       "href": "coaps://[2001:db8:a::b1d4]:44444/myLight",
576       "rt": ["oic.r.switch.binary"],
577       "if": ["oic.if.a", "oic.if.baseline"],
578       "p": {"bm": 3, "sec": true, "port": 44444}
579     },
580     {
581       "href": "coaps://[2001:db8:a::b1d4]:44444/oic/sec/doxm",
582       "rt": ["oic.r.doxm"],
583       "if": ["oic.if.baseline"],
584       "p": {"bm": 1, "sec": true, "port": 44444}
585     },
586     {
587       "href": "coaps://[2001:db8:a::b1d4]:44444/oic/sec/pstat",
588       "rt": ["oic.r.pstat"],
589       "if": ["oic.if.baseline"],
590       "p": {"bm": 1, "sec": true, "port": 44444}
591     },
592     {
593       "href": "coaps://[2001:db8:a::b1d4]:44444/oic/sec/cred",

```

```

594     "rt": ["oic.r.cred"],
595     "if": ["oic.if.baseline"],
596     "p": {"bm": 1, "sec": true, "port": 44444}
597   },
598   {
599     "href": "coaps://[2001:db8:a::b1d4]:44444/oic/sec/acl2",
600     "rt": ["oic.r.acl2"],
601     "if": ["oic.if.baseline"],
602     "p": {"bm": 1, "sec": true, "port": 44444}
603   },
604   {
605     "href": "coaps://[2001:db8:a::b1d4]:44444/myLightIntrospection",
606     "rt": ["oic.wk.introspection"],
607     "if": ["oic.if.r", "oic.if.baseline"],
608     "p": {"bm": 3, "sec": true, "port": 44444}
609   }
610 ]
611 }
612 ]

```

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

615
616 When an OCF Client requests a content format of “application/vnd.ocf+cbor”, the same bridge
617 will return information corresponding to the JSON below. (Note that what is returned is not in the
618 JSON format but in a suitable encoding as defined in the OCF 1.0 Core Specification.)

```

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

```

```

655 {
656   "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
657   "href": "/oic/sec/doxm",
658   "rt": ["oic.r.doxm"],
659   "if": ["oic.if.baseline"],
660   "p": {"bm": 1},
661   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
662 },
663 {
664   "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
665   "href": "/oic/sec/pstat",
666   "rt": ["oic.r.pstat"],
667   "if": ["oic.if.baseline"],
668   "p": {"bm": 1},
669   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
670 },
671 {
672   "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
673   "href": "/oic/sec/cred",
674   "rt": ["oic.r.cred"],
675   "if": ["oic.if.baseline"],
676   "p": {"bm": 1},
677   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
678 },
679 {
680   "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
681   "href": "/oic/sec/acl2",
682   "rt": ["oic.r.acl2"],
683   "if": ["oic.if.baseline"],
684   "p": {"bm": 1},
685   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
686 },
687 {
688   "anchor": "ocf://e61c3e6b-9c54-4b81-8ce5-f9039c1d04d9",
689   "href": "/myIntrospection",
690   "rt": ["oic.wk.introspection"],
691   "if": ["oic.if.r", "oic.if.baseline"],
692   "p": {"bm": 3},
693   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:11111"}]
694 },
695
696
697 {
698   "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
699   "href": "/oic/res",
700   "rt": ["oic.wk.res"],
701   "if": ["oic.if.ll", "oic.if.baseline"],
702   "p": {"bm": 3},
703   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
704 },
705 {
706   "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
707   "href": "/oic/d",
708   "rt": ["oic.wk.d", "oic.d.fan", "oic.d.virtual"],
709   "if": ["oic.if.r", "oic.if.baseline"],
710   "p": {"bm": 3},
711   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
712 },
713 {
714   "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
715   "href": "/oic/p",
716   "rt": ["oic.wk.p"],
717   "if": ["oic.if.r", "oic.if.baseline"],

```

```

718     "p": {"bm": 3},
719     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
720   },
721   {
722     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
723     "href": "/myFan",
724     "rt": ["oic.r.switch.binary"],
725     "if": ["oic.if.a", "oic.if.baseline"],
726     "p": {"bm": 3},
727     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
728   },
729   {
730     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
731     "href": "/oic/sec/doxm",
732     "rt": ["oic.r.doxm"],
733     "if": ["oic.if.baseline"],
734     "p": {"bm": 1},
735     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
736   },
737   {
738     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
739     "href": "/oic/sec/pstat",
740     "rt": ["oic.r.pstat"],
741     "if": ["oic.if.baseline"],
742     "p": {"bm": 1},
743     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
744   },
745   {
746     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
747     "href": "/oic/sec/cred",
748     "rt": ["oic.r.cred"],
749     "if": ["oic.if.baseline"],
750     "p": {"bm": 1},
751     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
752   },
753   {
754     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
755     "href": "/oic/sec/acl2",
756     "rt": ["oic.r.acl2"],
757     "if": ["oic.if.baseline"],
758     "p": {"bm": 1},
759     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
760   },
761   {
762     "anchor": "ocf://88b7c7f0-4b51-4e0a-9faa-cfb439fd7f49",
763     "href": "/myFanIntrospection",
764     "rt": ["oic.wk.introspection"],
765     "if": ["oic.if.r", "oic.if.baseline"],
766     "p": {"bm": 3},
767     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:22222"}]
768   },
769   {
770     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
771     "href": "/oic/res",
772     "rt": ["oic.wk.res"],
773     "if": ["oic.if.ll", "oic.if.baseline"],
774     "p": {"bm": 3},
775     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
776   },
777   {
778     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
779     "href": "/oic/d",

```

```

781     "rt": ["oic.wk.d", "oic.d.light", "oic.d.virtual"],
782     "if": ["oic.if.r", "oic.if.baseline"],
783     "p": {"bm": 3},
784     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
785   },
786   {
787     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
788     "href": "/oic/p",
789     "rt": ["oic.wk.p"],
790     "if": ["oic.if.r", "oic.if.baseline"],
791     "p": {"bm": 3},
792     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
793   },
794   {
795     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
796     "href": "/myLight",
797     "rt": ["oic.r.switch.binary"],
798     "if": ["oic.if.a", "oic.if.baseline"],
799     "p": {"bm": 3},
800     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
801   },
802   {
803     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
804     "href": "/oic/sec/doxm",
805     "rt": ["oic.r.doxm"],
806     "if": ["oic.if.baseline"],
807     "p": {"bm": 1},
808     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
809   },
810   {
811     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
812     "href": "/oic/sec/pstat",
813     "rt": ["oic.r.pstat"],
814     "if": ["oic.if.baseline"],
815     "p": {"bm": 1},
816     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
817   },
818   {
819     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
820     "href": "/oic/sec/cred",
821     "rt": ["oic.r.cred"],
822     "if": ["oic.if.baseline"],
823     "p": {"bm": 1},
824     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
825   },
826   {
827     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
828     "href": "/oic/sec/acl2",
829     "rt": ["oic.r.acl2"],
830     "if": ["oic.if.baseline"],
831     "p": {"bm": 1},
832     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
833   },
834   {
835     "anchor": "ocf://dc70373c-1e8d-4fb3-962e-017eaa863989",
836     "href": "/myLightIntrospection",
837     "rt": ["oic.wk.introspection"],
838     "if": ["oic.if.r", "oic.if.baseline"],
839     "p": {"bm": 3},
840     "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:33333"}]
841   },
842   {
843   {

```

```

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

```

```
907 {
908   "anchor": "ocf://8202138e-aa22-452c-b512-9ebad02bef7c",
909   "href": "/myLightIntrospection",
910   "rt": ["oic.wk.introspection"],
911   "if": ["oic.if.r", "oic.if.baseline"],
912   "p": {"bm": 3},
913   "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:44444"}]
914 }
915 ]
```

916 **6.3 General Requirements**

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

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

937 **6.4 Security**

938 **6.4.1 General Security Considerations**

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

942
943 The OCF Bridge Device shall be field updatable. (This requirement need not be tested but can
944 be certified via a vendor declaration.)

945
946 Unless an administrator opts in to allow it (see section 9.2), a translator shall not expose
947 connectivity to devices that it cannot get a secure connection to.

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

952 The Translator shall provide a “piid” value that can be used to correlate a non-OCF Device with its
953 corresponding Virtual OCF Device, as specified in Section 6.3. An Onboarding Tool might use this
954 correlation to improve the Onboarding user experience by eliminating or reducing the need for user
955 input, by automatically creating security settings for Virtual OCF Devices that are equivalent to the
956 security settings of their corresponding non-OCF Devices. See the OCF Security Specification for
957 detailed information about Onboarding.

958 Each Virtual Device shall implement the security requirements of the ecosystem that it is connected
959 to. For example, each Virtual OCF Device shall implement the behaviour required by the OCF 1.0

960 Core Specification and the OCF Security Specification. Each Virtual OCF Device shall perform
961 authentication, access control, and encryption according to the security settings it received from
962 the Onboarding Tool.

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

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

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

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

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

983 **6.4.2 Blocking communication of Bridged Devices with the OCF ecosystem**

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

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

991 **7 AllJoyn Translation**

992 **7.1 Requirements Specific to an AllJoyn Translator**

993 **7.1.1 Introduction**

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 **7.1.2 Exposing AllJoyn producer devices to OCF Clients**

1000 **7.1.2.1 Virtual OCF Devices and Resources**

1001 As specified in the OCF Security Specification, the value of the "di" property of OCF Devices
1002 (including Virtual OCF Devices) shall be established as part of Onboarding of that Virtual OCF
1003 Device.

1004
1005 Each AllJoyn object shall be mapped to one or more Virtual OCF Resources. If all AllJoyn
1006 interfaces can be translated to resource types on the same resource (as discussed below), there
1007 should be a single Virtual OCF Resource, and the path component of the URI of the Virtual OCF
1008 Resource shall be the AllJoyn object path. Otherwise, a Resource with that path shall exist with a
1009 Resource type of ["oic.wk.col", "oic.r.alljoynobject"] which is a Collection of links, where
1010 "oic.r.alljoynobject" is defined in Section 9.3, and the items in the collection are the Resources with
1011 the translated Resource Types as discussed below.

1012
1013 The value of the "piid" property of "/oic/d" for each Virtual OCF Device shall be the value of the
1014 OCF-defined AllJoyn field "org.openconnectivity.piid" in the AllJoyn About Announce signal, if that
1015 field exists, else it shall be calculated by the Translator as follows:

- 1016 • If the AllJoyn device supports security, the value of the "piid" property value shall be the
1017 peer GUID.
- 1018 • If the AllJoyn device does not support security but the device is being bridged anyway (see
1019 section 9.2), the "piid" property value shall be derived from the DeviceId and Appld
1020 properties (in the About data), by concatenating the DeviceId value (not including any null
1021 termination) and the Appld bytes and using the result as the "name" to be used in the
1022 algorithm specified in IETF RFC 4122 section 4.3, with SHA-1 as the hash algorithm, and
1023 8f0e4e90-79e5-11e6-bdf4-0800200c9a66 as the name space ID. (This is to address the
1024 problem of being able to de-duplicate AllJoyn devices exposed via separate OCF Bridge
1025 Devices.)
1026

1027 A translator implementation is encouraged to listen for AllJoyn About Announce signals matching
1028 any AllJoyn interface name. It can maintain a cache of information it received from these signals,
1029 and use the cache to quickly handle "/oic/res" queries from OCF Clients (without having to wait for
1030 Announce signals while handling the queries).

1031 A translator implementation is encouraged to listen for other signals (including
1032 EmitsChangedSignal of properties) only when there is a client subscribed to a corresponding
1033 resource on a Virtual AllJoyn Device.
1034

1035 There are multiple types of AllJoyn interfaces, which shall be handled as follows.

- 1036 • If the AllJoyn interface is in a well-defined set (defined in OCF ASA Mapping or section
1037 7.1.2.2 below) of interfaces where standard forms exist on both the AllJoyn and OCF
1038 sides, the translator shall either:
 - 1039 a. follow the specification for translating that interface specially, or
 - 1040 b. not translate the AllJoyn interface.
- 1041 • If the AllJoyn interface is not in the well-defined set, the translator shall either:
 - 1042 a. not translate the AllJoyn interface, or
 - 1043 b. algorithmically map the AllJoyn interface as specified in section 7.2 to
1044 custom/vendor-defined Resource Types by converting the AllJoyn interface
1045 name to OCF resource type name(s).
1046
1047

1048 An AllJoyn interface name shall be converted to a Device Type or a set of one or more OCF
1049 Resource Types as follows:

- 1050 1) If the AllJoyn interface has any members, append a suffix ".<seeBelow>" where <seeBelow>
1051 is described below.
- 1052 2) For each upper-case letter present in the entire string, replace it with a hyphen followed by
1053 the lower-case version of that letter (e.g., convert "A" to "-a").

- 1054 3) If an underscore appears followed by a (lower-case) letter or a hyphen, for each such
 1055 occurrence, replace the underscore with two hyphens (e.g., convert “_a” to “--a”, “_a” to
 1056 “---a”).
 1057 4) For each underscore remaining, replace it with a hyphen (e.g., convert “_1” to “-1”).
 1058 5) Prepend the “x.” prefix
 1059

1060 Some examples are shown in the table below. The first three are normal AllJoyn names
 1061 converted to unusual OCF names. The last three are unusual AllJoyn names converted
 1062 (perhaps back) to normal OCF names. (“xn--” is a normal domain name prefix for the
 1063 Punycode-encoded form of an Internationalized Domain Name, and hence can appear in a
 1064 normal vendor-specific OCF name.)
 1065

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

1066 Each AllJoyn interface that has members and is using algorithmic mapping shall be mapped to one
 1067 or more Resource Types as follows:
 1068

- 1069 • AllJoyn Properties with the same EmitsChangedSignal value are mapped to the same
 1070 Resource Type where the value of the <seeBelow> label is the value of
 1071 EmitsChangedSignal. AllJoyn Properties with EmitsChangedSignal values of “const” or
 1072 “false”, are mapped to Resources that are not Observable, whereas AllJoyn Properties with
 1073 EmitsChangedSignal values of “true” or “invalidates” result in Resources that are
 1074 Observable. The Version property in an AllJoyn interface is always considered “const”,
 1075 even if not specified in introspection XML.
- 1076 • Resource Types mapping AllJoyn Properties with access “readwrite” shall support the
 1077 “oic.if.rw” Interface. Resource Types mapping AllJoyn Properties with access “read” shall
 1078 support the “oic.if.r” Interface. Resource Types supporting both the “oic.if.rw” and “oic.if.r”
 1079 Interfaces shall choose “oic.if.r” as the default Interface.
- 1080 • Each AllJoyn Method is mapped to a separate Resource Type, where the value of the
 1081 <seeBelow> label is the AllJoyn Method name. The Resource Type shall support the
 1082 “oic.if.rw” Interface. Each argument of the AllJoyn Method is mapped to a separate
 1083 Property on the Resource Type, where the name of that Property is prefixed with the
 1084 AllJoyn Method name in order to help get uniqueness across all Resource Types on the
 1085 same Resource. When the AllJoyn argument name is not specified, the name of the
 1086 property shall be “x.<AllJoynInterfaceName>.<MethodName>arg<#>”, where
 1087 <AllJoynInterfaceName> is transformed as described above and <#> is the 0-indexed
 1088 position of the argument in the AllJoyn introspection XML, prefixed with the AllJoyn Method
 1089 name. In addition, that Resource Type has an extra
 1090 “x.<AllJoynInterfaceName>.<MethodName>validity” property that indicates whether the
 1091 rest of the properties have valid values. When the values are sent as part of an UPDATE
 1092 response, the validity property is true and any other properties have valid values. In a
 1093 RETRIEVE (GET or equivalent in the relevant transport binding) response, the validity
 1094 property is false and any other properties can have meaningless values. If the validity
 1095 property appears in an UPDATE request, its value shall be true (a value of false shall result
 1096 in an error response).
- 1097 • Each AllJoyn Signal (whether sessionless, sessioncast, or unicast) is mapped to a separate
 1098 Resource Type on an Observable Resource, where the value of the <seeBelow> label is

1099 the AllJoyn Signal name. The Resource Type shall support the "oic.if.r" Interface. Each
 1100 argument of the AllJoyn Signal is mapped to a separate Property on the Resource Type,
 1101 where the name of that Property is prefixed with the AllJoyn Signal name in order to help
 1102 get uniqueness across all Resource Types on the same Resource. When the AllJoyn
 1103 argument name is not specified, the name of the property shall be
 1104 "x.<AllJoynInterfaceName>.<SignalName>arg<#>", where <AllJoynInterfaceName> is
 1105 transformed as described above and <#> is the 0-indexed position of the argument in the
 1106 AllJoyn introspection XML, prefixed with the AllJoyn Signal name. In addition, that
 1107 Resource Type has an extra "x.<AllJoynInterfaceName>.<SignalName>validity" property
 1108 (also prefixed with the Signal name) that indicates whether the rest of the properties have
 1109 valid values. When the values are sent as part of a NOTIFY response, the validity property
 1110 is true and any other properties have valid values. In a RETRIEVE (GET or equivalent in
 1111 the relevant transport binding) response, the validity property is false and any other
 1112 properties returned can have meaningless values. This is because in AllJoyn, the signals
 1113 are instantaneous events and the values are not necessarily meaningful beyond the lifetime
 1114 of that message. Note that AllJoyn does have a TTL field that allows store-and-forward
 1115 signals, but such support is not required in OCF 1.0. We expect that in the future, the TTL
 1116 may be used to allow valid values in response to a RETRIEVE that is within the TTL.

1117 When an algorithmic mapping is used, AllJoyn data types shall be mapped to OCF property types
 1118 according to Section 7.2.
 1119

1120 If an AllJoyn operation fails, the translator shall send an appropriate OCF error response to the
 1121 OCF client. If an AllJoyn error name is available, it shall construct an appropriate OCF error
 1122 message (e.g., diagnostic payload if using CoAP) from the AllJoyn error name and AllJoyn error
 1123 message (if any), using the form "<error name>: <error message>". The <error name> shall be
 1124 taken from the AllJoyn error name field, with the "org.openconnectivity.Error." prefix removed if it
 1125 is present. If the resulting error name is of the form "<#>" where <#> is an error code without a
 1126 decimal (e.g., "404"), the error code shall be the error code indicated by the error name. Example:
 1127 "org.openconnectivity.Error.404" becomes "404", which shall result in an error 4.04 for a CoAP
 1128 transport.

1129 7.1.2.2 Exposing an AllJoyn producer application as a Virtual OCF Server

1130 Table 1 shows how OCF Device properties, as specified in Table 20 in the OCF 1.0 Core
 1131 Specification, shall be derived, typically from fields specified in the AllJoyn About Interface
 1132 Specification and AllJoyn Configuration Interface Specification.
 1133
 1134

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 For example, "Bob's Thermostat"	Y	AppName (no exact equivalent exists)	Application name assigned by the app manufacturer (developer or the OEM).	Y
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 Table 5 for DeviceID.	Y	(none)	Use as defined in the OCF Security Specification	

Protocol-Independent ID	piid	Unique identifier for OCF Device (UUID)	Y	<p>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.</p> <p>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.</p>	<p>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.</p> <p>DeviceId: Device identifier set by platform-specific means.</p> <p>Appld: A 128-bit globally unique identifier for the application. The Appld shall be a universally unique identifier as specified in IETF RFC 4122.</p>	<p>Peer GUID: conditionally Y</p> <p>DeviceId: Y</p> <p>Appld: Y</p>
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	<p>Comma separated list of the Version property values of each interface listed in the objectDescription argument of the Announce signal of About. Each value is formatted as "<interface name>.<Version property value>".</p>	<p>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.</p> <p>Note that AllJoyn specifies that the default value is 1 if the "org.gtk.GDBus.Since" annotation is absent.</p>	<p>N, but required by IRB for all standard interfaces, and absence can be used to imply a constant (e.g., 0)</p>
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

1135
1136
1137
1138
1139
1140
1141
1142
1143

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

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

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 which the device is located geographically. The free form text shall not start with a quote (").	N	org.openconnectivity.r (if it exists, else property shall be absent)		N
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	N	AppName	Application name assigned by the app manufacturer (developer or the OEM).	Y

		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.				
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

1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154

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, are 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 (UIUID); this shall be a UUID in accordance with IETF RFC 4122. It is recommended that the UUID be created using the random generation scheme (version 4 UUID) specific in the RFC.	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 IETF RFC 4122 section 4.3, with SHA-1 as the hash algorithm, and 8f0e4e90-79e5-11e6-bdf4-0800200c9a66 as the name space ID.	Name of the device set by platform-specific means (such as Linux and Android).	Y
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)	mnml	URL to manufacturer (not to exceed 32 characters)	N	org.openconnectivity.mnml (if it exists, else property shall be absent)		N
Model Number	mnmo	Model number as designated by manufacturer	N	ModelNumber	The app model number.	Y
Date of Manufacture	mndt	Manufacturing date of device	N	DateOfManufacture	Date of manufacture using format YYYY-MM-DD (known as XML DateTime format).	N
Platform Version	mnpv	Version of platform – string (defined by manufacturer)	N	org.openconnectivity.mnpv (if it exists, else property shall be absent)		N
OS Version	mnos	Version of platform resident OS – string (defined by manufacturer)	N	org.openconnectivity.mnos (if it exists, else property shall be absent)		N
Hardware Version	mnhw	Version of platform hardware	N	HardwareVersion	Hardware version of the device on which the app is running.	N
Firmware version	mnfv	Version of device firmware	N	org.openconnectivity.mnfv (if it exists, else property shall be absent)		N
Support URL	mnsi	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

1155
1156
1157
1158
1159

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

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	Mnpr	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.
--	--	--	--	--	--	-------------------------------------

1160
1161 In addition, the “oic.wk.mnt” properties Factory_Reset (“fr”) and Reboot (“rb”) shall be mapped to
1162 AllJoyn Configuration methods FactoryReset and Restart, respectively.

1163 **7.1.3 Exposing OCF resources to AllJoyn consumer applications**

1164 **7.1.3.1 Use of AllJoyn Producer Application**

1165 Unless specified otherwise, each OCF resource shall be mapped to a separate AllJoyn object.

1166
1167 Each OCF Server shall be exposed as a separate AllJoyn producer application, with its own About
1168 data. This allows platform-specific, device-specific, and resource-specific fields to all be
1169 preserved across translation. However, this requires that AllJoyn Claiming of such producer
1170 applications be solved in a way that does not require user interaction, but this is left as an
1171 implementation issue.

1172
1173 The AllJoyn producer application shall implement the “oic.d.virtual” AllJoyn interface. This allows
1174 translators to determine if a device is already being translated when multiple translators are
1175 present. The “oic.d.virtual” interface is defined as follows:

```
1176 <interface name="oic.d.virtual"/>
```

1177
1178
1179 The implementation may choose to implement this interface by the AllJoyn object at path “/oic/d”.

1180
1181 The AllJoyn peer ID shall be the OCF device ID (“di”).

1182
1183 Unless specified otherwise, the AllJoyn object path on the resource shall be the OCF URI path.
1184 The AllJoyn About data shall be populated per Table 5 below.

1185
1186 A translator implementation is encouraged to maintain a cache of OCF resources to handle
1187 WholImplements queries from the AllJoyn side, and emit an Announce Signal for each OCF Server.
1188 Specifically, the translator could always Observe “/oic/res” changes and only Observe other
1189 resources when there is a client with a session on a Virtual AllJoyn Device.

1190
1191 There are multiple types of resources, which shall be handled as follows.

- 1192 • If the Resource Type is in a well-defined set (defined in OCF ASA Mapping or section
1193 7.1.3.2 below) of resource types where standard forms exist on both the AllJoyn and OCF
1194 sides, the translator shall either:
 - 1195 a. follow the specification for translating that resource type specially, or
 - 1196 b. not translate the Resource Type.
- 1197 • If the Resource Type is not in the well-defined set (but is not a Device Type), the translator
1198 shall either:
 - 1199 a. not translate the Resource Type, or
 - 1200 b. algorithmically map the Resource Type as specified in section 7.2 to a
1201 custom/vendor-defined AllJoyn interface by converting the OCF Resource Type
1202 name to an AllJoyn Interface name.

1203
1204 An OCF Resource Type or Device Type name shall be converted to an AllJoyn interface name as
1205 follows:

- 1206 1) Remove the “x.” prefix if present
- 1207 2) For each occurrence of a hyphen (in order from left to right in the string):

- 1208 a. If the hyphen is followed by a letter, replace both characters with a single upper-
 1209 case version of that letter (e.g., convert “-a” to “A”).
 1210 b. Else, if the hyphen is followed by another hyphen followed by either a letter or a
 1211 hyphen, replace two hyphens with a single underscore (e.g., convert “--a” to “_a”).
 1212 c. Else, convert the hyphen to an underscore (i.e., convert “-” to “_”).
 1213

1214 Some examples are shown in the table below. The first three are unusual OCF names
 1215 converted (perhaps back) to normal AllJoyn names. The last three are normal OCF names
 1216 converted to unusual AllJoyn names. (“xn--” is a normal domain name prefix for the Punycode-
 1217 encoded form of an Internationalized Domain Name, and hence can appear in a normal vendor-
 1218 specific OCF name.)
 1219

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

1220
 1221 An OCF Device Type is mapped to an AllJoyn interface with no members.
 1222

1223 Unless specified otherwise, each OCF Resource Type shall be mapped to an AllJoyn interface as
 1224 follows:

- 1225 • Each OCF property is mapped to an AllJoyn property in that interface.
- 1226 • The EmitsChangedSignal value for each AllJoyn property shall be set to “true” if the
 1227 resource supports NOTIFY, or “false” if it does not. (The value is never set to “const” or
 1228 “invalidates” since those concepts cannot currently be expressed in OCF.)
- 1229 • The “access” attribute for each AllJoyn property shall be “read” if the OCF property is read-
 1230 only, or “readwrite” if the OCF property is read-write.
- 1231 • If the resource supports DELETE, a Delete() method shall appear in the interface.
- 1232 • If the resource supports CREATE, a Create() method shall appear in the interface, with
 1233 input arguments of each property of the resource to create. (Such information is not
 1234 available algorithmically in OIC 1.1 but can be determined in OCF 1.0 via introspection.) If
 1235 such information is not available, a CreateWithDefaultValues() method shall appear which
 1236 takes no input arguments. In either case, the output argument shall be an OBJECT_PATH
 1237 containing the path of the created resource.
- 1238 • If the resource supports UPDATE (i.e., the “oic.if.rw” or “oic.if,a” interface) then an AllJoyn
 1239 property set operation (i.e., an org.freedesktop.DBus.Properties.Set() method call) shall be
 1240 mapped to a Partial UPDATE (e.g., POST in CoAP) with the corresponding OCF property.
- 1241 • If a Resource has a Resource Type “oic.r.alljoynobject”, then instead of separately
 1242 translating each of the Resources in the collection to its own AllJoyn object, all Resources
 1243 in the collection shall instead be translated to a single AllJoyn object whose object path is
 1244 the OCF URI path of the collection.

1245 OCF property types shall be mapped to AllJoyn data types according to Section 7.2.
 1246

1247 If an OCF operation fails, the translator shall send an appropriate AllJoyn error response to the
 1248 AllJoyn consumer. If an error message is present in the OCF response, and the error message
 1249 (e.g., diagnostic payload if using CoAP) fits the pattern “<error name>: <error message>” where
 1250 <error name> conforms to the AllJoyn error name syntax requirements, the error name and error
 1251 message shall be extracted from the error message. Otherwise, the error name shall be
 1252 “org.openconnectivity.Error.<#>” where <#> is the error code without a decimal (e.g., “404”).

1253

1254 **7.1.3.2 Exposing an OCF server as a Virtual AllJoyn Producer**1255 Table 5 shows how AllJoyn About Interface fields are derived, based on properties in “oic.wk.d”,
1256 “oic.wk.con”, “oic.wk.p”, or “oic.wk.con.p”.

1257

1258

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. For example, [{"language": "en", "value": "Dave's Laptop"}]	N
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	N (ln), Y (n)

To AJ Field name	AJ Description	AJ Mand?	From OCF Property title	OCF Property name	OCF Description	OCF Mand?
					Name (n) value shall be included in this array.	
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 used by the application.	Y	(none)		Translator should return its own value	
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	mnsI	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

To AJ Field name	AJ Description	AJ Mand?	From OCF Property title	OCF Property name	OCF Description	OCF Mand?
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

1259
1260
1261
1262
1263
1264
1265
1266
1267
1268

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” or “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

To AJ Field name	AJ Description	AJ Mand?	From OCF Property title	OCF Property name	OCF Description	OCF Mand?
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

1269
1270
1271
1272
1273
1274
1275
1276

In addition, the Configuration methods FactoryReset and Restart shall be mapped to "oic.wk.mnt" properties Factory_Reset ("fr") and Reboot ("rb"), respectively, and any additional 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) shall be mapped to vendor-defined fields the AllJoyn Configuration data, with a field name formed by removing the leading "x." from the property name.

1277 7.2 On-the-Fly Translation from D-Bus and OCF payloads

1278 7.2.1 Introduction

1279 The "dbus1" payload format is specified in the D-Bus Specification and AllJoyn adopted the D-Bus
1280 protocol and made it distributed over the network. The modifications done by AllJoyn to the format are all
1281 in the header part of the packet, not in the data payload itself, which remains compatible with "dbus1". Other
1282 variants of the protocol that have been proposed by the Linux community ("GVariant" and "kdbus" payloads)
1283 contain slight incompatibilities and are not relevant for this discussion.

1284 7.2.2 Translation without aid of introspection

1285 7.2.2.1 Introduction

1286 This section describes how translators shall translate messages between the two payload formats in the
1287 absence of introspection metadata from the actual device. This situation arises in the following cases:

- 1288 • Requests to OIC 1.1 devices
- 1289 • Replies from OIC 1.1 devices
- 1290 • Content not described by introspection, such as the inner payload of AllJoyn properties of type
1291 "D-Bus VARIANT".

1292 Since introspection is not available, the translator cannot know the rich JSON sub-type, only the underlying
1293 CBOR type and from that it can infer the JSON generic type, and hence translation is specified below in
1294 terms of those generic types.

1295 **7.2.2.2 Booleans**

1296 Boolean conversion is trivial since both sides support this type.

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

1297 **7.2.2.3 Numeric types**

1298 The translation of numeric types is lossy and that is unavoidable due to the limited expressiveness
1299 of the JSON generic types. This can only be solved with introspection.

1300 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)	
"i" - INT32 (signed 32-bit)	
"x" - INT64 (signed 64-bit) ⁽¹⁾	
"d" - DOUBLE (IEEE 754 double precision)	

1301

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

1302

1303 Notes and rationales:

- 1304 1. D-Bus payloads of types "t" (UINT64) and "x" (INT64) can contain values that cannot be perfectly
1305 represented in IEEE 754 double-precision floating point. The RFCs governing JSON do not forbid
1306 such numbers but caution that many implementations may not be able to deal with them.
1307 Currently, OCF transports its payload using CBOR instead of JSON, which can represent those
1308 numbers with fidelity. However, it should be noted that the OCF 1.0 Core Specification does
1309 not allow for integral numbers outside the range $-2^{53} \leq x \leq 2^{53}$.
- 1310 2. To provide the most predictable result, all translations from OCF to AllJoyn produce values of
1311 type "d" DOUBLE (IEEE 754 double precision).

1312 **7.2.2.4 Text strings**

D-Bus type	JSON type
"s" - STRING	String

1313

1314 Conversion between D-Bus and JSON strings is simple, as both require their content to be valid Unicode.
1315 For example, an implementation can typically do a direct byte copy, as both protocols specify UTF-8 as
1316 the encoding of the data, neither constrains the data to a given normalisation format nor specify whether
1317 private-use characters or non-characters should be disallowed.

1318 Since the length of D-Bus strings is always known, it is recommended translators not use CBOR
1319 indeterminate text strings (first byte 0x7f).

1320 7.2.2.5 Byte arrays

1321 The translation of a byte array is direction-specific.

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

1322

1323 The base64url encoding is specified in IETF RFC 4648 section 5.

1324 7.2.2.6 D-Bus Variants

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

1325

1326 D-Bus has a type called VARIANT ("v") that is a wrapper around any other D-Bus type. It's a way for the
1327 type system to perform type-erasure. JSON, on the other hand, is not type-safe, which means that all
1328 JSON values are, technically, variants. The conversion for a D-Bus variant to JSON is performed by
1329 entering that variant and encoding the type carried inside as per the rules in this document.

1330 The algorithm must be recursive, as D-Bus variants are allowed to contain variants themselves.

1331 7.2.2.7 D-Bus Object Paths and Signatures

1332 The translation of D-Bus object paths and signatures is unidirectional (there is no mapping *to* them,
1333 only *from* them). In the reverse direction, Section 7.2.2.4 always converts to D-Bus STRING
1334 rather than OBJECT_PATH or SIGNATURE since it is assumed that "s" is the most common string
1335 type in use.

From D-Bus type	To JSON type
"o" - OBJECT_PATH	String
"g" - SIGNATURE	

1336

1337 Both D-Bus object paths and D-Bus type signatures are US-ASCII strings with specific formation rules,
1338 found in the D-Bus Specification. They are very seldom used and are not expected to be found in
1339 resources subject to translation without the aid of introspection.

1340 7.2.2.8 D-Bus Structures

1341 The translation of the following types is direction-specific:

From D-Bus type	To JSON type
"r" - STRUCT	array, length > 0

1342

1343 D-Bus structures can be interpreted as a fixed-length array containing a pre-determined list of types for
1344 each member. This is how such a structure is mapped to JSON: as an array of heterogeneous content,
1345 which are the exact members of the D-Bus structure, in the order in which they appear in the structure.

1346 7.2.2.9 Arrays

1347 The translation of the following types is bidirectional:

D-Bus type	JSON type
“ay” - ARRAY of BYTE	(base64-encoded) string – see Section 7.2.2.5
“ae” - ARRAY of DICT_ENTRY	object – see Section 7.2.2.10

1348

1349 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

1350

1351

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

1352

1353 Aside from arrays of bytes and arrays of dictionary entries, which are mapped to JSON strings and objects
1354 respectively, arrays in JSON cannot be constrained to a single type (i.e., heterogeneous arrays). For that
1355 reason, strictly speaking all D-Bus arrays excepting arrays of bytes and arrays of dictionary entries must
1356 first be converted to arrays of variant “av” and then that array can be converted to JSON.

1357 Conversion of D-Bus arrays of variants uses the conversion of variants as specified above, which simply
1358 eliminates the distinction between a variant containing a given value and that value outside a variant. In
1359 other words, the elements of a D-Bus array are extracted and sent as elements of the JSON array, as per
1360 the other rules of this document.

1361 7.2.2.10 Dictionaries / Objects

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

1362

1363 The choice of “dictionary of STRING to VARIANT” is made because that is the most common type of
 1364 dictionary found in payloads and is an almost perfect superset of all possible dictionaries in D-Bus anyway.
 1365 Moreover, it can represent JSON Objects with fidelity, which is the representation that OCF uses in its data
 1366 models, which in turn means those D-Bus dictionaries will be able to carry with fidelity any OCF JSON
 1367 Object in current use.

1368 D-Bus dictionaries that are not mapping string to variant are first converted to those constraints and then
 1369 encoded in CBOR.

1370 **7.2.2.11 Non-translatable types**

D-Bus Type
“h” – UNIX_FD (Unix file descriptor)

JSON type
null
undefined (not officially valid JSON, but some implementations permit it)

1371
 1372 The above types are not translatable, and the translator should drop the incoming message. None of the
 1373 types above are in current use by either AllJoyn, OIC 1.1, or future OCF 1.0 devices, so the inability to
 1374 translate them should not be a problem.

1375 **7.2.2.12 Examples**

1376

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

Source D-Bus	JSON Result
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”

1377

1378

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”)

Source JSON	D-Bus Result
[]	ARRAY<VARIANT>()
[1]	ARRAY<IDOUBLE>(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" → DOUBLE(1.0))
{"1": 1}	map<STRING, VARIANT>("1" → DOUBLE(1.0))
{ "rep": { "state": false, "power": 1.0, "name": "My Light" } }	map<STRING, VARIANT>({ "rep", map<STRING, VARIANT>({ "state", BOOLEAN(FALSE)}, "power", DOUBLE(1.0)}, "name", STRING("My Light")})

1379
1380
1381
1382
1383

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.

1384 **7.2.3 Translation with aid of introspection**

1385 **7.2.3.1 Introduction to Introspection Metadata**

1386 When introspection is available, the translator can use the extra metadata provided by the side offering the
1387 service to expose a higher-quality reply to the other side. This chapter details modifications to the translation
1388 described in the previous chapter when the metadata is found.

1389 Introspection metadata can be used for both translating requests to services and replies from those services.
1390 When used to translate requests, the introspection is “constraining”, since the translator must conform
1391 exactly to what that service expects. When used to translate replies, the introspection is “relaxing”, but may
1392 be used to inform the receiver what other possible values may be encountered in the future.

1393 Note that OCF introspection uses JSON types, media attributes, and format attributes, not CBOR encoding.
1394 The actual encoding of each JSON type is discussed in Section 12.3 of the OCF 1.0 Core Specification,
1395 JSON format attribute values are as defined in JSON Schema Validation, and JSON media attribute
1396 values are as defined in JSON Hyper-Schema.

1397 **7.2.3.2 Translation of the introspection itself**

1398 Note that both OCF 1.0 and AllJoyn require all services exposed to include introspection metadata, which
1399 means the translator will need to translate the introspection information on-the-fly for each OCF resource
1400 or AllJoyn producer it finds. The translator shall preserve as much of the original information as can be
1401 represented in the translated format. This includes both the information used in machine interactions and
1402 the information used in user interactions, such as description and documentation text.

1403 **7.2.3.3 Variability of introspection data**

1404 Introspection data is not a constant and the translator may find, upon discovering further services, that the
 1405 D-Bus interface or OCF Resource Type it had previously encountered is different than previously seen. The
 1406 translator needs to take care about how the destination side will react to a change in introspection.

1407 D-Bus interfaces used by AllJoyn services may be updated to newer versions, which means a given type
 1408 of service may be offered by two distinct versions of the same interface. Updates to standardised interfaces
 1409 must follow strict guidelines established by the AllSeen Interface Review Board, mapping each version to
 1410 a different OCF Resource Type should be possible without much difficulty. However, there’s no guarantee
 1411 that vendor-specific extensions follow those requirements. Indeed, there’s nothing preventing two revisions
 1412 of a product to contain completely incompatible interfaces that have the same name and version number.

1413 On the opposite direction, the rules are much laxer. Since OCF specifies optional properties to its Resource
 1414 Types, a simple monotonically-increasing version number like AllJoyn consumer applications expect is not
 1415 possible.

1416 However, it should be noted that services created by the translator by “on-the-fly” translation will only be
 1417 accessed by generic client applications. Dedicated applications will only use “deep binding” translation.

1418 **7.2.3.4 Numeric types**

1419 For numeric values, all D-Bus and JSON numeric types are treated equally as source and may all be
 1420 translated into any of the other side’s types. When translating a request to a service, the translator need
 1421 only verify whether there would be loss of information when translating from source to destination. For
 1422 example, when translating the number 1.5 to either a JSON integer or to one of the D-Bus integral types,
 1423 there would be loss of information, in which case the translator should refuse the incoming message.
 1424 Similarly, the value 1,234,567 does not fit the range of a D-Bus byte, 16-bit signed or unsigned integer.

1425 When translating the reply from the service, the translator shall use the following rules.

1426 The following table indicates how to translate from a JSON type to the corresponding D-Bus type, where
 1427 the first matching row shall be used. If the JSON schema does not indicate the minimum value of a JSON
 1428 integer, 0 is the default. If the JSON schema does not indicate the maximum value of a JSON integer, 2^{32}
 1429 – 1 is the default. The resulting AllJoyn introspection XML shall contain “org.alljoyn.Bus.Type.Min” and
 1430 “org.alljoyn.Bus.Type.Max” annotations whenever the minimum or maximum, respectively, of the JSON
 1431 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)

1432

1433 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 ≤ 2 ⁵³ , else string with JSON format attribute “uint64”	IETF RFC 7159 section 6 explains that higher JSON integers are not interoperable.
“x” (INT64)	integer (if org.alljoyn.Bus.Type.Min ≥ -2 ⁵³ AND org.alljoyn.Bus.Type.Max ≤ 2 ⁵³), else string with JSON format attribute “int64”	IETF RFC 7159 section 6 explains that other JSON integers are not interoperable.
“d” (double)	number	

1434

1435 **7.2.3.5 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

1436

1437 There’s no difference in the translation of text strings and byte arrays compared to the previous section.

1438 This section simply lists the JSON equivalent types for the generated OCF introspection.

1439 In addition, the mapping of the following JSON Types is direction-specific:

From JSON type	Condition	To D-Bus Type
String	pattern = “^[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}\$”	“ay” – ARRAY of BYTE

1440

1441 JSON strings with any other format value (e.g., date-time, uri, etc.) or pattern value not shown in this table
1442 above shall be treated the same as if the format and pattern attributes were absent, by simply mapping
1443 the value to a D-Bus string.

1444 7.2.3.6 D-Bus Variants

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

1445
1446 If the introspection of an AllJoyn producer indicates a value in a request should be a D-Bus VARIANT, the
1447 translator should create such a variant and encode the incoming value as the variant's payload as per the
1448 rules in the rest of this document.

1449 7.2.3.7 D-Bus Object Paths and Signatures

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

1450
1451 If the introspection of an AllJoyn producer indicates a value in a request should be a D-Bus Object Path or
1452 D-Bus Signature, the translator should perform a validity check in the incoming CBOR Text String. If the
1453 incoming data fails to pass this check, the message should be rejected.

1454 7.2.3.8 D-Bus Structures

1455 D-Bus structure members are described in the introspection XML with the
1456 "org.alljoyn.Bus.Struct.*StructureName*.Field.*fieldName*.Type" annotation. The translator shall use
1457 the AJSoftwareVersion field of the About data obtained from a bridged AllJoyn producer as follows.
1458 When the version of AllJoyn implemented on the Bridged Device is v16.10.00 or greater and the
1459 member annotations are present, the translator shall use a JSON object to represent a structure,
1460 mapping each member to the entry with that name. The translator needs to be aware that the
1461 incoming CBOR payload may have changed the order of the fields, when compared to the D-Bus
1462 structure. When the version of AllJoyn implemented on the Bridged Device is less than v16.10.00,
1463 the translator shall follow the rule for translating D-Bus structures without the aid of introspection
1464 data.

1465 7.2.3.9 Arrays and Dictionaries

1466 If the introspection of the AllJoyn interface indicates that the array is neither an ARRAY of BYTE ("ay") nor
1467 an ARRAY of VARIANT ("av") or that the dictionary is not mapping STRING to VARIANT ("a{sv}"), the
1468 translator shall apply the constraining or relaxing rules specified in other sections.

1469 Similarly, if the OCF introspection indicates a homogeneous array type, the information about the array's
1470 element type should be used as the D-Bus array type instead of VARIANT ("v").

1471 7.2.3.10 Other JSON format attribute values

1472 The JSON format attribute may include other custom attribute types. They are not known at this time, but
1473 it is expected that those types be handled by their type and representation alone.

7.2.3.11 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
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 = value
ARRAY<INT32>()		[]	JSON schema should indicate: type = array items = integer
ARRAY<INT64>()		[]	JSON schema should indicate: type = array items = string items.pattern = ^0 ([1-9][0-9]{0,18})\$
STRUCT< INT32, INT32>(0, 1)	AllJoyn introspection specifies the argument with the annotation:	["x": 0, "y": 1]	JSON schema should indicate: type = object element.x.type = integer

AllJoyn Source	AllJoyn Introspection Notes	Translated JSON Payload	OCF Introspection Notes
)	<pre><struct name="Point"> <field name="x" type="i"/> <field name="y" type="i"/> </struct></pre>		element.y.type = integer

1475

CBOR Payload	OCF Introspection Notes	Translated AllJoyn	AllJoyn Introspection Notes
0	JSON type is "integer"	INT32(0)	
0	JSON type is "integer" minimum = -2^{40} maximum = 2^{40}	INT64(0)	org.alljoyn.Bus.Type.Min = -2^{40} org.alljoyn.Bus.Type.Max = 2^{40}
0	JSON type is "integer" minimum = 0 maximum = 2^{48}	UINT64(0)	org.alljoyn.Bus.Type.Max = 2^{48}
0.0	JSON type is "float"	DOUBLE(0.0)	
[1]	JSON schema indicates: type = array items = integer element.minimum = 0 element.maximum = 2^{46}	ARRAY<UINT64>(1)	org.alljoyn.Bus.Type.Max = 2^{46}

1476 **8 Device Type Definitions**

1477 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

1478 9 Resource Type definitions

1479 9.1 List of resource types

1480 All the sections in 9 describe the Resource Types with a restful API definition language. The
1481 Resource Type definitions presented in 9 are formatted for readability, and so may appear to have
1482 extra line breaks. The contents of the Resource Types without the extra line breaks are available
1483 in OCF Resource Type Definitions.

1484

1485

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

1486

1487 9.2 Secure Mode

1488 9.2.1 Introduction

1489 This resource describes a secure mode on/off feature (on/off).

1490 A secureMode value of “true” means that the feature is on, and:

- 1491 • any Bridged Server that cannot be communicated with securely shall not have a
1492 corresponding Virtual OCF Server, and
- 1493 • any Bridged Client that cannot be communicated with securely shall not have a
1494 corresponding Virtual OCF Client.

1495 A secureMode value of “false” means that the feature is off, and:

- 1496 • any Bridged Server can have a corresponding Virtual OCF Server, and
- 1497 • any Bridged Client can have a corresponding Virtual OCF Client.

1498 9.2.2 Example URI Path

1499 /example/SecureModeResURI

1500 9.2.3 Resource Type

1501 The resource type (rt) is defined as: oic.r.securemode.

1502 9.2.4 RAML Definition

```
1503 #%RAML 0.8
1504 title: OCFSecureMode
1505 version: v1.0.0-20170531
1506 traits:
1507   - interface:
1508     queryParameters:
1509       if:
1510         enum: ["oic.if.rw", "oic.if.baseline"]
```

1511

1512 /example/SecureModeResURI:

```
1513   description: |
```

```

1514     This resource describes a secure mode on/off feature (on/off).
1515     A secureMode value of "true" means that the feature is on, and any Bridged Server that cannot
1516     be communicated with securely shall not have a corresponding Virtual OCF Server, and any Bridged
1517     Client that cannot be communicated with securely shall not have a corresponding Virtual OCF Client.
1518     A secureMode value of "false" means that the feature is off, any Bridged Server can have a
1519     corresponding Virtual OCF Server, and any Bridged Client can have a corresponding Virtual OCF
1520     Client.
1521
1522     is: ['interface']
1523
1524     get:
1525         description: |
1526             Retrieves the value of secureMode.
1527         responses:
1528             200:
1529                 body:
1530                     application/json:
1531                         schema: /
1532                             {
1533                                 "id": "https://www.openconnectivity.org/ocf-
1534     apis/bridging/schemas/oic.r.securemode.json#",
1535                                 "$schema": "http://json-schema.org/draft-04/schema#",
1536                                 "description" : "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
1537     reserved.",
1538                                 "title": "Secure Mode",
1539                                 "definitions": {
1540                                     "oic.r.securemode": {
1541                                         "type": "object",
1542                                         "properties": {
1543                                             "secureMode": {
1544                                                 "type": "boolean",
1545                                                 "description": "Status of the Secure Mode"
1546                                             }
1547                                         }
1548                                     },
1549                                 },
1550                                 "type": "object",
1551                                 "allOf": [
1552                                     {"$ref": "../core/schemas/oic.core-schema.json#/definitions/oic.core"},
1553                                     {"$ref": "#/definitions/oic.r.securemode"}
1554                                 ],
1555                                 "required": [ "secureMode" ]
1556                             }
1557                         example: /
1558                             {
1559                                 "rt":          ["oic.r.securemode"],
1560                                 "id":          "unique_example_id",
1561                                 "secureMode": false
1562                             }
1563
1564     post:
1565         description: |
1566             Updates the value of secureMode.
1567         body:
1568             application/json:
1569                 schema: /
1570                     {
1571                         "id": "https://www.openconnectivity.org/ocf-
1572     apis/bridging/schemas/oic.r.securemode.json#",
1573                         "$schema": "http://json-schema.org/draft-04/schema#",
1574                         "description" : "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
1575     reserved.",
1576                         "title": "Secure Mode",

```

```

1577     "definitions": {
1578       "oic.r.securemode": {
1579         "type": "object",
1580         "properties": {
1581           "secureMode": {
1582             "type": "boolean",
1583             "description": "Status of the Secure Mode"
1584           }
1585         }
1586       }
1587     },
1588     "type": "object",
1589     "allOf": [
1590       {"$ref": "../core/schemas/oic.core-schema.json#/definitions/oic.core"},
1591       {"$ref": "#/definitions/oic.r.securemode"}
1592     ],
1593     "required": [ "secureMode" ]
1594   }
1595
1596   example: /
1597     {
1598       "id":          "unique_example_id",
1599       "secureMode": true
1600     }
1601
1602   responses:
1603     200:
1604       body:
1605         application/json:
1606           schema: /
1607             {
1608               "id": "https://www.openconnectivity.org/ocf-
apis/bridging/schemas/oic.r.securemode.json#",
1609               "$schema": "http://json-schema.org/draft-04/schema#",
1610               "description": "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
reserved.",
1611               "title": "Secure Mode",
1612               "definitions": {
1613                 "oic.r.securemode": {
1614                   "type": "object",
1615                   "properties": {
1616                     "secureMode": {
1617                       "type": "boolean",
1618                       "description": "Status of the Secure Mode"
1619                     }
1620                 }
1621               }
1622             },
1623             "type": "object",
1624             "allOf": [
1625               {"$ref": "../core/schemas/oic.core-schema.json#/definitions/oic.core"},
1626               {"$ref": "#/definitions/oic.r.securemode"}
1627             ],
1628             "required": [ "secureMode" ]
1629           }
1630
1631           example: /
1632             {
1633               "id":          "unique_example_id",
1634               "secureMode": true
1635             }
1636

```

1637 9.2.5 Swagger2.0 Definition

```

1638 {
1639   "swagger": "2.0",
1640   "info": {
1641     "title": "OCFSecureMode",

```

```

1642     "version": "v1.0.0-20170531",
1643     "license": {
1644         "name": "copyright 2016-2017 Open Connectivity Foundation, Inc. All rights reserved.",
1645         "x-description": "Redistribution and use in source and binary forms, with or without
1646 modification, are permitted provided that the following conditions are met:\n      1.
1647 Redistributions of source code must retain the above copyright notice, this list of conditions and
1648 the following disclaimer.\n      2. Redistributions in binary form must reproduce the above
1649 copyright notice, this list of conditions and the following disclaimer in the documentation and/or
1650 other materials provided with the distribution.\n\n      THIS SOFTWARE IS PROVIDED BY THE Open
1651 Connectivity Foundation, INC. \ "AS IS\ " AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
1652 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR
1653 WARRANTIES OF NON-INFRINGEMENT, ARE DISCLAIMED.\n      IN NO EVENT SHALL THE Open Connectivity
1654 Foundation, INC. OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
1655 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
1656 OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)\n      HOWEVER CAUSED AND
1657 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
1658 OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
1659 OF SUCH DAMAGE.\n"
1660     }
1661 },
1662 "schemes": ["http"],
1663 "consumes": ["application/json"],
1664 "produces": ["application/json"],
1665 "paths": {
1666     "/example/SecureModeResURI" : {
1667         "get": {
1668             "description": "This resource describes a secure mode on/off feature (on/off).\nA
1669 secureMode value of 'true' means that the feature is on, and any Bridged Server that cannot be
1670 communicated with securely shall not have a corresponding Virtual OCF Server, and any Bridged
1671 Client that cannot be communicated with securely shall not have a corresponding Virtual OCF
1672 Client.\nA secureMode value of 'false' means that the feature is off, any Bridged Server can have a
1673 corresponding Virtual OCF Server, and any Bridged Client can have a corresponding Virtual OCF
1674 Client.\nRetrieves the value of secureMode.\n",
1675             "parameters": [
1676                 { "$ref": "#/parameters/interface" }
1677             ],
1678             "responses": {
1679                 "200": {
1680                     "description": "",
1681                     "x-example":
1682                     {
1683                         "rt": ["oic.r.securemode"],
1684                         "id": "unique_example_id",
1685                         "secureMode": false
1686                     }
1687                 },
1688                 "schema": { "$ref": "#/definitions/SecureMode" }
1689             }
1690         }
1691     },
1692     "post": {
1693         "description": "Updates the value of secureMode.\n",
1694         "parameters": [
1695             { "$ref": "#/parameters/interface" },
1696             {
1697                 "name": "body",
1698                 "in": "body",
1699                 "required": true,
1700                 "schema": { "$ref": "#/definitions/SecureMode" },
1701                 "x-example":
1702                 {
1703                     "id": "unique_example_id",
1704                     "secureMode": true
1705                 }
1706             }
1707         ],
1708         "responses": {
1709             "200": {
1710                 "description": "",
1711                 "x-example":
1712                 {

```

```

1713         "id": "unique_example_id",
1714         "secureMode": true
1715     }
1716 },
1717     "schema": { "$ref": "#/definitions/SecureMode" }
1718 }
1719 }
1720 }
1721 },
1722 },
1723 "parameters": {
1724     "interface": {
1725         "in": "query",
1726         "name": "if",
1727         "type": "string",
1728         "enum": ["oic.if.rw", "oic.if.baseline"]
1729     }
1730 },
1731 "definitions": {
1732     "SecureMode": {
1733         {
1734         "properties": {
1735             "id": {
1736                 "description": "Instance ID of this specific resource",
1737                 "maxLength": 64,
1738                 "readOnly": true,
1739                 "type": "string"
1740             },
1741             "if": {
1742                 "description": "The interface set supported by this resource",
1743                 "items": {
1744                     "enum": [
1745                         "oic.if.baseline",
1746                         "oic.if.ll",
1747                         "oic.if.b",
1748                         "oic.if.lb",
1749                         "oic.if.rw",
1750                         "oic.if.r",
1751                         "oic.if.a",
1752                         "oic.if.s"
1753                     ],
1754                     "type": "string"
1755                 },
1756                 "minItems": 1,
1757                 "readOnly": true,
1758                 "type": "array"
1759             },
1760             "n": {
1761                 "description": "Friendly name of the resource",
1762                 "maxLength": 64,
1763                 "readOnly": true,
1764                 "type": "string"
1765             },
1766             "rt": {
1767                 "description": "Resource Type",
1768                 "items": {
1769                     "maxLength": 64,
1770                     "type": "string"
1771                 },
1772                 "minItems": 1,
1773                 "readOnly": true,
1774                 "type": "array"
1775             },
1776             "secureMode": {
1777                 "description": "Status of the Secure Mode",
1778                 "type": "boolean"
1779             }
1780         },
1781         "required": [
1782             "secureMode"
1783         ],

```

```

1784         "type": "object"
1785     }
1786 }
1787 }
1788 }
1789 }
1790

```

1791 9.2.6 Property Definition

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

1792 9.2.7 CRUDN behaviour

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

1793

1794 9.3 AllJoyn Object

1795 9.3.1 Introduction

1796 This resource is a collection of resources that were all derived from the same AllJoyn object.

1797 9.3.2 Example URI Path

1798 /example/AllJoynObject/

1799 9.3.3 Resource Type

1800 The resource type (rt) is defined as: oic.r.alljoynobject.

1801 9.3.4 RAML Definition

```

1802 #%RAML 0.8
1803 title: OCFAllJoynObject
1804 version: v1.0.0-20170531
1805 traits:
1806   - interface-baseline:
1807     queryParameters:
1808       if:
1809         enum: ["oic.if.baseline"]
1810   - interface-ll:
1811     queryParameters:
1812       if:
1813         enum: ["oic.if.ll"]
1814
1815 /example/AllJoynObject/?if=oic.if.baseline:
1816   description: |
1817     This resource is a collection of resources that were all derived from the same AllJoyn object.
1818
1819   is: ['interface-baseline']
1820
1821   get:
1822     description: |
1823       Retrieves the current AllJoyn object information.
1824     responses:
1825       200:
1826         body:
1827           application/json:

```

```

1828     schema: /
1829         {
1830             "id": "https://www.openconnectivity.org/ocf-
1831 apis/bridging/schemas/oic.r.alljoynobject.json#",
1832             "$schema": "http://json-schema.org/draft-04/schema#",
1833             "description" : "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
1834 reserved.",
1835             "title": "AllJoyn Object",
1836             "definitions": {
1837                 "oic.r.alljoynobject": {
1838                     "type": "object",
1839                     "allOf": [
1840                         {
1841                             "$ref": "../../core/schemas/oic.collection-
1842 schema.json#/definitions/oic.collection"
1843                         },
1844                         {
1845                             "properties": {
1846                                 "rt": {
1847                                     "type": "array",
1848                                     "minItems": 2,
1849                                     "maxItems": 2,
1850                                     "uniqueItems": true,
1851                                     "items": {
1852                                         "enum": ["oic.r.alljoynobject", "oic.wk.col"]
1853                                     }
1854                                 }
1855                             }
1856                         }
1857                     ]
1858                 },
1859             },
1860             "type": "object",
1861             "allOf": [
1862                 {"$ref": "../../core/schemas/oic.core-schema.json#/definitions/oic.core"},
1863                 {"$ref": "#/definitions/oic.r.alljoynobject"}
1864             ]
1865         }
1866     example: /
1867         {
1868             "rt":      ["oic.r.alljoynobject", "oic.wk.col"],
1869             "id":      "unique_example_id",
1870             "links": [
1871                 {"href": "/myRes1URI", "rt": ["x.example.widget.false"], "if":
1872 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1873                 {"href": "/myRes2URI", "rt": ["x.example.widget.true"], "if":
1874 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1875                 {"href": "/myRes3URI", "rt": ["x.example.widget.method1"], "if":
1876 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1877                 {"href": "/myRes4URI", "rt": ["x.example.widget.method2"], "if":
1878 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]}
1879             ]
1880         }
1881 /example/AllJoynObject/?if=oic.if.ll:
1882     description: |
1883         This resource is a collection of resources that were all derived from the same AllJoyn object.
1884
1885     is: ['interface-ll']
1886
1887     get:
1888     description: |
1889         Retrieves the Links in the current AllJoyn object.
1890
1891     responses:
1892         200:
1893             body:

```

```

1893     application/json:
1894         schema: /
1895             {
1896                 "id": "https://www.openconnectivity.org/ocf-
1897 apis/bridging/schemas/oic.r.alljoynobject-11#",
1898                 "$schema": "http://json-schema.org/draft-04/schema#",
1899                 "description": "Copyright (c) 2017 Open Connectivity Foundation, Inc. All rights
1900 reserved.",
1901                 "title": "AllJoyn Object Links List Schema",
1902                 "definitions": {
1903                     "oic.r.alljoynobject-11": {
1904                         "allOf": [
1905                             {
1906                                 "$ref": "../../core/schemas/oic.collection.linkslist-
1907 schema.json#/definitions/oic.collection.alllinks"
1908                             }
1909                         ]
1910                     },
1911                 },
1912                 "allOf": [
1913                     {"$ref": "#/definitions/oic.r.alljoynobject-11"}
1914                 ]
1915             }
1916         example: /
1917             [
1918                 {"href": "/myRes1URI", "rt": ["x.example.widget.false"], "if":
1919 ["oic.if.r","oic.if.rw","oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a:b1d4]:1111"}]},
1920                 {"href": "/myRes2URI", "rt": ["x.example.widget.true"], "if":
1921 ["oic.if.r","oic.if.rw","oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a:b1d4]:1111"}]},
1922                 {"href": "/myRes3URI", "rt": ["x.example.widget.method1"], "if":
1923 ["oic.if.rw","oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a:b1d4]:1111"}]},
1924                 {"href": "/myRes4URI", "rt": ["x.example.widget.method2"], "if":
1925 ["oic.if.rw","oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a:b1d4]:1111"}]}
1926             ]
1927
1928

```

1929 9.3.5 Swagger2.0 Definition

```

1930 {
1931     "swagger": "2.0",
1932     "info": {
1933         "title": "OCFAllJoynObject",
1934         "version": "v1.0.0-20170531",
1935         "license": {
1936             "name": "copyright 2016-2017 Open Connectivity Foundation, Inc. All rights reserved.",
1937             "x-description": "Redistribution and use in source and binary forms, with or without
1938 modification, are permitted provided that the following conditions are met:\n
1939 1. Redistributions of source code must retain the above copyright notice, this list of conditions and
1940 the following disclaimer.\n
1941 2. Redistributions in binary form must reproduce the above
1942 copyright notice, this list of conditions and the following disclaimer in the documentation and/or
1943 other materials provided with the distribution.\n\n
1944 THIS SOFTWARE IS PROVIDED BY THE Open
1945 Connectivity Foundation, INC. \AS IS\ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
1946 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE OR
1947 WARRANTIES OF NON-INFRINGEMENT, ARE DISCLAIMED.\n
1948 IN NO EVENT SHALL THE Open Connectivity
1949 Foundation, INC. OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
1950 EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
1951 OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)\n
1952 HOWEVER CAUSED AND
1953 ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
1954 OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
1955 OF SUCH DAMAGE.\n"
1956         }
1957     },
1958     "schemes": ["http"],
1959     "consumes": ["application/json"],
1960     "produces": ["application/json"],
1961     "paths": {
1962         "/example/AllJoynObject/?if=oic.if.11" : {
1963             "get": {

```

```

1960         "description": "This resource is a collection of resources that were all derived from the
1961 same AllJoyn object.\nRetrieves the Links in the current AllJoyn object.\n",
1962         "parameters": [
1963             {"$ref": "#/parameters/interface-11"}
1964         ],
1965         "responses": {
1966             "200": {
1967                 "description": "",
1968                 "x-example":
1969                 [
1970                     {"href": "/myRes1URI", "rt": ["x.example.widget.false"], "if":
1971 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1972                     {"href": "/myRes2URI", "rt": ["x.example.widget.true"], "if":
1973 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1974                     {"href": "/myRes3URI", "rt": ["x.example.widget.method1"], "if":
1975 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
1976                     {"href": "/myRes4URI", "rt": ["x.example.widget.method2"], "if":
1977 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]}
1978                 ]
1979             },
1980             "schema": { "$ref": "#/definitions/AllJoynObject-11" }
1981         }
1982     }
1983 }
1984 },
1985 "/example/AllJoynObject/?if=oic.if.baseline" : {
1986     "get": {
1987         "description": "This resource is a collection of resources that were all derived from the
1988 same AllJoyn object.\nRetrieves the current AllJoyn object information.\n",
1989         "parameters": [
1990             {"$ref": "#/parameters/interface-baseline"}
1991         ],
1992         "responses": {
1993             "200": {
1994                 "description": "",
1995                 "x-example":
1996                 {
1997                     "rt": ["oic.r.alljoynobject", "oic.wk.col"],
1998                     "id": "unique_example_id",
1999                     "links": [
2000                         {"href": "/myRes1URI", "rt": ["x.example.widget.false"], "if":
2001 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
2002                         {"href": "/myRes2URI", "rt": ["x.example.widget.true"], "if":
2003 ["oic.if.r", "oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
2004                         {"href": "/myRes3URI", "rt": ["x.example.widget.method1"], "if":
2005 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]},
2006                         {"href": "/myRes4URI", "rt": ["x.example.widget.method2"], "if":
2007 ["oic.if.rw", "oic.if.baseline"], "eps": [{"ep": "coaps://[2001:db8:a::b1d4]:1111"}]}
2008                     ]
2009                 }
2010             },
2011             "schema": { "$ref": "#/definitions/AllJoynObject" }
2012         }
2013     }
2014 }
2015 }
2016 },
2017 "parameters": {
2018     "interface-11" : {
2019         "in" : "query",
2020         "name" : "if",
2021         "type" : "string",
2022         "enum" : ["oic.if.11"]
2023     },
2024     "interface-baseline" : {
2025         "in" : "query",
2026         "name" : "if",
2027         "type" : "string",
2028         "enum" : ["oic.if.baseline"]
2029     }
2030 },

```

```

2031 "definitions": {
2032   "AllJoynObject-ll" :
2033     {
2034       "allOf": [
2035         {
2036           "$ref": "../../core/schemas/oic.collection.linkslist-schema.json"
2037         }
2038       ]
2039     }
2040
2041   ,
2042   "AllJoynObject" :
2043     {
2044       "allOf": [
2045         {
2046           "$ref": "../../core/schemas/oic.collection-schema.json"
2047         },
2048         {
2049           "properties": {
2050             "id": {
2051               "description": "Instance ID of this specific resource",
2052               "maxLength": 64,
2053               "readOnly": true,
2054               "type": "string"
2055             },
2056             "if": {
2057               "description": "The interface set supported by this resource",
2058               "items": {
2059                 "enum": [
2060                   "oic.if.baseline",
2061                   "oic.if.ll",
2062                   "oic.if.b",
2063                   "oic.if.lb",
2064                   "oic.if.rw",
2065                   "oic.if.r",
2066                   "oic.if.a",
2067                   "oic.if.s"
2068                 ],
2069                 "type": "string"
2070               },
2071               "minItems": 1,
2072               "readOnly": true,
2073               "type": "array"
2074             },
2075             "n": {
2076               "description": "Friendly name of the resource",
2077               "maxLength": 64,
2078               "readOnly": true,
2079               "type": "string"
2080             },
2081             "rt": {
2082               "items": {
2083                 "enum": [
2084                   "oic.r.alljoynobject",
2085                   "oic.wk.col"
2086                 ]
2087               },
2088               "maxItems": 2,
2089               "minItems": 2,
2090               "type": "array",
2091               "uniqueItems": true
2092             }
2093           }
2094         }
2095       ],
2096       "type": "object"
2097     }
2098   }
2099 }
2100 }
2101

```

2102

2103 **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

2104