



Document Identifier: DSP0271

Date: 2016-10-13

Version: 0.5.6

1
2
3
4
5

6 YANG to Redfish Mapping Specification

Information for Work-in-Progress version:

IMPORTANT: This document is not a standard. It does not necessarily reflect the views of the DMTF or its members. Because this document is a Work in Progress, this document may still change, perhaps profoundly and without notice. This document is available for public review and comment until superseded.

Provide any comments through the DMTF Feedback Portal:

<http://www.dmtf.org/standards/feedback>

7 **Supersedes: None**
8 **Document Class: Normative**
9 **Document Status: Work in Progress**
10 **Document Language: en-US**

11

12 Copyright Notice

13 Copyright © 2016 Distributed Management Task Force, Inc. (DMTF). All rights reserved.

14 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
15 management and interoperability. Members and non-members may reproduce DMTF specifications and
16 documents, provided that correct attribution is given. As DMTF specifications may be revised from time to
17 time, the particular version and release date should always be noted.

18 Implementation of certain elements of this standard or proposed standard may be subject to third party
19 patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations
20 to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose,
21 or identify any or all such third party patent right, owners or claimants, nor for any incomplete or
22 inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to
23 any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize,
24 disclose, or identify any such third party patent rights, or for such party's reliance on the standard or
25 incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any
26 party implementing such standard, whether such implementation is foreseeable or not, nor to any patent
27 owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is
28 withdrawn or modified after publication, and shall be indemnified and held harmless by any party
29 implementing the standard from any and all claims of infringement by a patent owner for such
30 implementations.

31 For information about patents held by third-parties which have notified the DMTF that, in their opinion,
32 such patent may relate to or impact implementations of DMTF standards, visit
33 <http://www.dmtf.org/about/policies/disclosures.php>.

34 This document's normative language is English. Translation into other languages is permitted.

CONTENTS

36	Foreword	6
37	Acknowledgments	6
38	Introduction.....	7
39	1 Scope	8
40	2 Normative references	8
41	3 Terms and definitions	8
42	4 Symbols and abbreviated terms.....	9
43	5 Description	9
44	5.1 YANG	9
45	5.2 Redfish.....	10
46	5.3 Differences between YANG and Redfish CSDL	10
47	5.3.1 Other mapping decisions	11
48	5.3.2 YANG namespace	11
49	5.3.3 Synthesized names for CSDL.....	11
50	5.3.4 OData annotations	11
51	5.4 Redfish resource URI.....	11
52	6 YANG statement mapping format	13
53	6.1 Module statement	15
54	6.1.1 Mapping YANG depiction to Redfish mockup	15
55	6.1.2 Mapping YANG code to Redfish CSDL	15
56	6.1.3 Import statement	17
57	6.1.4 Include statement.....	18
58	6.1.5 Namespace statement.....	19
59	6.1.6 Prefix statement.....	19
60	6.2 Submodule statement	19
61	6.2.1 Belongs-to statement	21
62	6.3 Typedef statement	21
63	6.3.1 Mapping YANG code to Redfish CSDL	21
64	6.3.2 Default statement.....	22
65	6.4 Type statement	22
66	6.4.1 Mapping YANG code to Redfish CSDL	23
67	6.4.2 Path statement.....	24
68	6.4.3 require-instance statement	24
69	6.4.4 Mapping special types	25
70	6.5 Container statement.....	28
71	6.5.1 Mapping the YANG depiction to Redfish mockup	28
72	6.5.2 Mapping YANG code to Redfish CSDL	29
73	6.6 Leaf statement	31
74	6.6.1 Mapping YANG depiction to Redfish mockup	31
75	6.6.2 Mapping YANG code to Redfish CSDL	31
76	6.7 Leaf-list statement.....	32
77	6.7.1 Mapping YANG depiction to Redfish mockup	32
78	6.7.2 Mapping YANG code to Redfish CSDL	32
79	6.8 List statement.....	33
80	6.8.1 Mapping YANG depiction to Redfish mockup	33
81	6.8.2 Mapping YANG code to Redfish CSDL	34
82	6.8.3 Key statement	36
83	6.9 Choice statement	36
84	6.9.1 Mapping the YANG depiction to Redfish mockup	37
85	6.9.2 Translating the YANG depiction to Redfish mockup	37
86	6.9.3 Case.....	39
87	6.10 Anyxml statement	39

88	6.10.1 Mapping YANG depiction to Redfish mockup	39
89	6.10.2 Mapping YANG code to Redfish CSDL	40
90	6.11 Grouping statement	40
91	6.12 Uses statement	41
92	6.12.1 Refine statement	42
93	6.13 Rpc statement	42
94	6.13.1 Mapping YANG code to Redfish CSDL	42
95	6.13.2 Input statement	43
96	6.13.3 Output statement	43
97	6.14 Notification statement	44
98	6.14.1 Mapping YANG code to Redfish CSDL	44
99	6.15 Augment statement	46
100	6.16 Identity statement	47
101	6.16.1 Mapping YANG code to Redfish CSDL	47
102	6.16.2 Base statement	48
103	6.17 Extension statement	48
104	6.18 Argument statement	49
105	6.19 Feature statement	50
106	6.20 If-feature statement	50
107	6.21 Deviation statement	51
108	6.22 Deviate statement	52
109	6.23 Config statement	52
110	6.24 Status statement	53
111	6.25 Description statement	53
112	6.26 Reference statement	54
113	6.27 When statement	54
114	6.28 Unmapped YANG statements	55
115	ANNEX A (informative) Change log	57
116		

117 **Tables**

118	Table 1 – Differences between YANG and Redfish	10
119	Table 2 - YANG statements	13
120	Table 3 – Module statement mapping	17
121	Table 4 – Import statement mapping	18
122	Table 5 – Submodule statement mapping	20
123	Table 6 – Typedef statement mapping	21
124	Table 7 – Built in YANG types	22
125	Table 8 – Type statement mapping	23
126	Table 9 – Container statement mapping	30
127	Table 10 – Leaf statement mapping	31
128	Table 11 – Leaf-list statement mapping	33
129	Table 12 – List statement mapping	35
130	Table 13 – Choice statement mapping	39
131	Table 14 – Anyxml statement mapping	40
132	Table 15 – Grouping statement mapping	41
133	Table 16 – Uses statement mapping	42
134	Table 17 – Rpc statement mapping	43
135	Table 18 – Notification statement mapping	45
136	Table 19 – Augment statement mapping	47

137 Table 20 – Identity statement mapping 48
138 Table 21 – Extension statement mapping..... 49
139 Table 22 – Argument statement mapping 50
140 Table 23 – Feature statement mapping 50
141 Table 24 – Deviation statement mapping 52
142 Table 25 – Deviate statement mapping 52
143
144
145

146

Foreword

147 The *YANG to Redfish Mapping Specification* was prepared by the Chinook Technical Working Group.

148 DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems
149 management and interoperability. For information about the DMTF, see <http://www.dmtf.org>.

150 Acknowledgments

151 The DMTF acknowledges the following individuals for their contributions to this document:

- 152 • Mittika Ganguli – Intel Corporation
- 153 • Ajay Gummadi - Microsoft
- 154 • Mike Lazar – Dell, Inc
- 155 • John Leung – Intel Corporation
- 156 • Peter Mellquist – Hewlett Packard Enterprise
- 157 • Michael Pizzo - Microsoft
- 158 • Rohan Sen - VMware
- 159 • Joseph White – Dell, Inc

160

Introduction

161 The information in this specification should be sufficient to convert a YANG model to a file which adheres
162 to the Common Schema Data Language (CSDL) format. CSDL is one of the formats that Redfish's uses
163 to describe schema and is described in OASIS OData specification (odata.org). The conversion can be
164 done manually or programmatically.

165

YANG to Redfish Mapping Specification

166 1 Scope

167 The *YANG to Redfish Mapping Specification* describes how to map a YANG model to a Redfish model,
168 specifically, the mapping to YANG RFCs to Redfish CSDLs.

169 The mapping should be universal enough to convert any YANG model. This will allow network devices to
170 be managed via the Redfish RESTful interface, regardless of the YANG model they support.

171 The specification uses IETF RFC 6020 as the description of the YANG model elements. The
172 specification uses examples from DHCP for usages of the YANG model elements.

173 This document describes a mapping translation. The goal is for completeness. However, there may be
174 YANG model elements and constructs beyond RFC 6020 which may need to be added.

175 2 Normative references

176 The following referenced documents are indispensable for the application of this document. For dated or
177 versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies.
178 For references without a date or version, the latest published edition of the referenced document
179 (including any corrigenda or DMTF update versions) applies.

180 DMTF DSP0266, "Redfish Scalable Platforms Management API Specification",
181 <http://www.dmtf.org/standards/redfish>

182 IETF RFC 6020, "YANG - A Data Modeling Language for the Network Configuration Protocol
183 (NETCONF)" <https://tools.ietf.org/html/rfc6020>

184 ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards,
185 <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype>

186 OASIS OData v4.0, <https://www.oasis-open.org/standards#odatav4.0>

187 3 Terms and definitions

188 In this document, some terms have a specific meaning beyond the normal English meaning. Those terms
189 are defined in this clause.

190 The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"),
191 "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described
192 in [ISO/IEC Directives, Part 2](#), Annex H. The terms in parentheses are alternatives for the preceding term,
193 for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that
194 [ISO/IEC Directives, Part 2](#), Annex H specifies additional alternatives. Occurrences of such additional
195 alternatives shall be interpreted in their normal English meaning.

196 The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as
197 described in [ISO/IEC Directives, Part 2](#), Clause 5.

198 The terms "normative" and "informative" in this document are to be interpreted as described in [ISO/IEC](#)
 199 [Directives, Part 2](#), Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do
 200 not contain normative content. Notes and examples are always informative elements.

201 The terms defined in [DSP0004](#), [DSP0223](#), and [DSP1001](#) apply to this document. The following additional
 202 terms are used in this document.

203 4 Symbols and abbreviated terms

204 The abbreviations defined in [DSP0004](#), [DSP0223](#), and [DSP1001](#) apply to this document. The following
 205 additional abbreviations are used in this document.

206 5 Description

207 This YANG to Redfish Mapping Guidelines document describes how to map YANG statements into
 208 Redfish OData CSDL constructs.

209 5.1 YANG

210 YANG is a data modeling language used to model configuration and state data manipulated by the
 211 Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF
 212 notifications. YANG is used to model the operations and content layers of NETCONF.

213 Various SDO have YANG RFCs for various network capabilities.

214 A YANG RFC includes a YANG depiction of the model (tree diagram) and YANG code (or schema). The
 215 YANG code is consider more normative than the YANG depiction.

216 The YANG depiction gives a high level view of the model's construct. Below is a fragment of the
 217 depiction from the DHCP Draft.

```
218     +--rw interfaces
219     |   +--rw interface* [name]
220     |     +--rw name                string
221     |     +--rw description?       string
222     |     +--rw type                identityref
223     |     +--rw enabled?           boolean
224     . . .
```

225 The YANG code specifies the schema associated with the YANG depiction. The YANG code is bracketed
 226 by <CODE BEGINS> and <CODE ENDS) delimiters. Below is a fragment of the YANG code.

```
227 <CODE BEGINS>
228 module ietf-interfaces {
229
230     namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces";
231     prefix if;
232
233     import ietf-yang-types {
234         prefix yang;
235     }
236
237     organization
238         "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
239
240     Contact "...";
241
```

242
243
244
245

```

container interfaces {
  description "Interface configuration parameters.";
  . . .
}
    
```

<CODE ENDS>

246 **5.2 Redfish**

247 The Redfish Scalable Platforms Management API ("Redfish") is a new specification that uses RESTful
248 interface semantics to access data defined in model format to perform systems management. It is suitable
249 for a wide range of servers, from stand-alone servers to rack mount and bladed environments but scales
250 equally well for large scale cloud environments.

251 RESTful interface specified by:

- 252 • A URI path to resource
- 253 • The content of the resource are described in an OData schema (CSDL) and json-schema

254 **5.3 Differences between YANG and Redfish CSDL**

255 There are basic differences between YANG and Redfish CSDL which are evident throughout. Table 1
256 contains systemic differences between YANG RFCs and Redfish CSDL. The table includes the decision
257 made for mapping purposes.

258 **Table 1 – Differences between YANG and Redfish**

YANG	Redfish JSON and CSDL	Mapping Decision
Names contain "-" (dashes)	OData does not allow dashes	Convert dashes to "_" underscore, when used in an identifier
Names contain ":" (colons)	OData does not allow colons	Convert colon to "." (period), when used in an identifier
Names are generally Camel case, but exceptions exist	Names are Pascal case	Use YANG naming
Some names are concatenations (e.g. dhcp/relay/dhcpRelayServerGroups)	Prefers shorter names (e.g. dhcp/relay/ServerGroups)	Use YANG naming
Container names are plural	URI uses plural forms (Systems), but CSDL use singular form ("SystemCollection")	Use YANG naming
YANG has implicit scoping based on containment	CSDL has explicit scoping based on namespaces	Synthesize names to retain YANG scoping
Containers may contain no leafs/properties	"Resources should contain properties (otherwise, consider eliminating resource)"	Include resources without properties
List nodes may have leafs/properties	Resource.Collections don't have properties	Place properties in a subordinate-resource
Reference statements are not normative	LongDescription properties contain normative text	Place LongDescription at the module level which normatively refers to the RFC

259 5.3.1 Other mapping decisions

260 These general decisions were also followed for the mapping the YANG models:

- 261 • Map RFCs as-is. Suppress the desire to optimize for CSDL
- 262 • Define everything in the schema and don't worry about feature exposure exclusion
- 263 • A YANG module will correspond to an entity type at the top level
- 264 • Treat YANG some statements as a pre-processor style directive (e.g., uses, grouping)

265 5.3.2 YANG namespace

266 Preserve the YANG naming, including case and spelling (e.g., module, node structure).

267 The above rule strays from the Redfish's Pascal-case capitalization convention, since most YANG RFCs
 268 use camel-case. The deviation is necessary to allow the YANG community to understand the resultant
 269 mapping collateral.

270 5.3.3 Synthesized names for CSDL

271 Some model translations will require synthesized names for intermediate objects in the CSDL version.
 272 The intent is to create a translated mapping such that the resulting derived schema and JSON message
 273 match what would be expected from reading the YANG model directly.

274 5.3.4 OData annotations

275 Liberal use of CSDL Annotations to encapsulate YANG model information.

276 For each YANG statement, an annotation shall exist which retains the value from the original YANG
 277 statement. For example, the *default* statement results in an annotation of Term="Redfish.Yang.default"
 278 and whose String attribute have the value of the <default value>, "enable".

```
279     default: "enable"
280
281     <Annotation Term="Redfish.Yang.default" String="enable"/>
```

282 If a YANG statement is specifies a YANG node, an annotation is added which specifies the type of node
 283 which the YANG statement specifies. YANG nodes exist for *module*, *submodule*, *container*, and *list*. For
 284 example, the following module statement results in the following annotation in the CSDL

```
285     module: ietf-system
286
287     <Annotation Term="Redfish.Yang.NodeName" String="ietf_system" >
288       <Annotation Term="Redfish.Yang.NodeType"
289         EnumMember ="Redfish.Yang.NodeTypes/module"/>
290     /Annotation>
```

291 If the value of YANG statement has double quotes, then the CSDL escaping rules should be follow in
 292 creating the annotation string.

293 5.4 Redfish resource URI

294 The resource which represents the YANG model is attached to the instance of the NetworkDevice.
 295 Because of the abundance of YANG definitions, the resource name is constructed from the organization
 296 and the module name.

```
297
298     ./NetworkDevices/{id}/<org>_<module-name>
```

299 The resource name is "ietf_interface" for IETF RFC 7317 (System) as shown below.

300

```
301 ./NetworkDevices/{id}/ietf_system
```

302 An example mockup of the NetworkDevice singleton resource is shown below. The properties for DHCP,
303 DNS and interfaces are shown.

```
304 {  
305     "@Redfish.Copyright": "",  
306     "@odata.context": "/redfish/v1/$metadata#NetworkDevices/Members/$entity",  
307     "@odata.type": "#NetworkDevice.v1_0_0.NetworkDevice",  
308     "@odata.id": "/redfish/v1/NetworkDevices/SW_15",  
309     "Id": "SW_15",  
310     "Name": "Ethernet Switch",  
311     "Manufacturer": "Manufacturer Name",  
312     "Model": "Model Name",  
313     "SKU": "67B",  
314     "SerialNumber": "2M220100SL",  
315     "PartNumber": "76-88883",  
316     "Dhcp": { "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp" },  
317     "Dns": { "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dns" },  
318     "Interfaces": { "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_interfaces" },  
319     "Links": {  
320         "Chassis": [{  
321             "@odata.id": "/redfish/v1/Chassis/NetworkDeviceChassis_1"  
322         }],  
323         "ManagedBy": [{  
324             "@odata.id": "/redfish/v1/Managers/NetworkDeviceManager_1"  
325         }]  
326     },  
327     "Actions": {  
328         "#NetworkDevice.Reset": {  
329             "target": "/redfish/v1/NetworkDevices/SW_15/Actions/NetworkDevice.Reset",  
330             "ResetType@Redfish.AllowableValues": [  
331                 "On",  
332                 "ForceOff",  
333                 "GracefulShutdown",  
334                 "ForceRestart"  
335             ]  
336         }  
337     }  
338 }  
339 }  
340 }  
341 }  
342 }  
343 }
```

344 **6 YANG statement mapping format**

345 This clause describes how the mapping is formatted which the remainder of this document.

346 The clauses follow the ordering from RFC6020. For each YANG statement, the clause will contain the
 347 three sub-clauses

- 348 • Mapping YANG Depiction to Redfish Mockup
- 349 • Mapping YANG code to Redfish CSDL
- 350 • Statement Mapping Table

351 The "Mapping YANG Depiction to Redfish Mockup" clause shows an example of how the YANG depiction
 352 would look as a Redfish mockup, if the mapping rules are followed. The Redfish mockup shows what the
 353 end-user will see, without looking at the schema. If a statement does not have a depiction, then this
 354 section may not exist.

355 The "Mapping YANG code to Redfish CSDL" specifies a mapping ruleset to convert YANG code to a
 356 model with adheres to the Redfish specification.

357 The Statement Mapping table contains the mapping rules for the statement and each allowable sub-
 358 statement. These tables are heavily cross-referenced. There are sub-sections for sub-statements for
 359 which additional text is beneficial to understanding the mapping.

360 Table 2 shows the set of YANG statements that will to be mapped in Redfish CSDL. The ordering of
 361 these statements mirrored the ordering in RFC6020.

362 Note: Uses and grouping statement should be resolved during the translation. Annotations as still added
 363 to retain the notion of uses/grouping relationship. The text in the Description column are taken from
 364 RFC6020.

365 **Table 2 - YANG statements**

YANG	Description	Details
module	The "module" statement defines the module's name, and groups all statements that belong to the module together.	Clause 6.1
submodule	The "submodule" statement defines the submodule's name, and groups all statements that belong to the submodule together.	Clause 6.2
typedef	The "typedef" statement defines a new type that may be used locally in the module, in modules or submodules which include it, and by other modules that import from it.	Clause 6.3
type	The "type" statement takes as an argument a string that is the name of a YANG built-in type or a derived type, followed by an optional block of sub-statements that are used to put further restrictions on the type.	Clause 6.4
container	The "container" statement is used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (including leafs, lists, containers, and leaf-lists).	Clause 6.5
leaf	The "leaf" statement contains simple data like an integer or a string. It has exactly one value of a particular type and no child nodes.	Clause 6.6
leaf-list	The "leaf-list" is a sequence of leaf nodes with exactly one value of a particular type per leaf.	Clause 6.7
list	The "list" statement defines a sequence of list entries.	Clause 6.8

YANG	Description	Details
choice	The "choice" statement defines a set of alternatives, only one of which may exist at any one time.	Clause 6.9
anyxml	The "anyxml" statement defines an interior node in the schema tree. The "anyxml" statement is used to represent an unknown chunk of XML.	Clause 6.10
grouping	The "grouping" statement is used to define a reusable block of nodes, which may be used locally in the module, in modules that include it, and by other modules that import from it.	Clause 6.11
uses	The "uses" statement is used to reference a "grouping" definition. It takes one argument, which is the name of the grouping.	Clause 6.12
rpc	The "rpc" statement is used to define a NETCONF RPC operation.	Clause 6.13
notification	The "notification" statement is used to define a NETCONF notification.	Clause 6.14
augment	The "augment" statement allows a module or submodule to add to the schema tree defined in an external module, or the current module and its submodules, and to add to the nodes from a grouping in a "uses" statement.	Clause 6.15
identity	The "identity" statement is used to define a new globally unique, abstract, and untyped identity.	Clause 6.16
extension	The "extension" statement allows the definition of new statements within the YANG language. This new statement definition can be imported and used by other modules.	Clause 6.17
argument	The "argument" statement, which is optional, takes as an argument a string that is the name of the argument to the keyword. If no argument statement is present, the keyword expects no argument when it is used.	Clause 6.18
feature	The "feature" statement is used to define a mechanism by which portions of the schema are marked as conditional. A feature name is defined that can later be referenced using the "if-feature" statement.	Clause 6.19
if-feature	The "if-feature" statement makes its parent statement conditional.	Clause 6.20
deviation	The "deviation" statement defines a hierarchy of a module that the device does not implement faithfully.	Clause 6.21
config	The "config" statement takes as an argument the string "true" or "false". If "config" is "true", the definition represents configuration.	Clause 6.22
status	The "status" statement takes as an argument one of the strings "current", "deprecated", or "obsolete".	Clause 6.24
description	The "description" statement takes as an argument a string that contains a human-readable textual description of this definition. The text is provided in a language (or languages) chosen by the module developer;	Clause 6.25
reference	The "reference" statement takes as an argument a string that is used to specify a textual cross-reference to an external document, either another module that defines related management information, or a document that provides additional information relevant to this definition.	Clause 6.26
when	The "when" statement makes its parent data definition statement conditional. The node defined by the parent data definition statement is only valid when the condition specified by the "when" statement is satisfied.	Clause 6.27

366 6.1 Module statement

367 From RFC6020, the "module" statement defines the module's name, and groups all statements that
 368 belong to the module together. The "module" statement's argument is the name of the module, followed
 369 by a block of sub-statements that hold detailed module information.

370 6.1.1 Mapping YANG depiction to Redfish mockup

371 The *module* statement is depicted as follows:

```
372 module: [module-name]
373
374 module: ietf-system (System example)
```

375 The resultant URI for the module resource is shown below. The module resource is a subordinate
 376 resource to the NetworkDevice resource.

377 In which, [modified-module-name] is synthesized by changing the dashes "-" to underscores "_" in the
 378 module-name.

```
379 ./NetworkDevices/{id}/[modified-module-name]
380
381 ./NetworkDevices/{id}/ietf_system (System example)
```

382 A mockup of the ietf_system resource is shown below.

```
383 {
384   "@Redfish.Copyright": "",
385
386   "@odata.context": "/redfish/v1/$metadata#NetworkDevices/Members/ietf_dhcp/$entity",
387   "@odata.type": "#ietf_dhcp.1.0.0.ietf_dhcp",
388   "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_system",
389
390   "Id": "ietf system",
391   "Name": "System",
392
393   "system": {
394     "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/system"
395   }
396   "system state": {
397     "@odata.id": "/redfish/v1/NetworkDevices/SW 15/ietf_dhcp/system state"
398   }
399 }
```

400 6.1.2 Mapping YANG code to Redfish CSDL

401 The YANG code for a *module* statement is shown below.

```
402 <CODE BEGINS> file "ietf-system@2014-08-06.yang"
403
404 module ietf-system {
405   namespace "urn:ietf:params:xml:ns:yang:ietf-system";
406   prefix "sys";
407
408   import ietf-yang-types {
409     prefix yang;
410   }
411
412   organization "IETF NETMOD (NETCONF Data Modeling Language) Working Group";
413   contact "...";
414   description "...";
415   revision "2014-12-18";
416   . . .
417 }
```

418 The resultant CSDL fragment is shown below. Note the following items in the mapping:

- 419 • The <edm:Reference> tag is constructed from the *import* statements. The Uri and Namespace
420 attributes are constructed from the *import* statement. The Alias attribute is constructed from the
421 *prefix* statement.
- 422 • The <Schema> tag is constructed from the *namespace* and *prefix* statements. The un-
423 versioned <Schema> tag uses the *prefix* statement.
- 424 • There is an annotation for Redfish.Yang.NodeType
- 425 • Three annotation are added to the *contact*, *description* and *revision* statements
- 426 • The annotations Redfish.Yang.description and Odata.Description are both present
427
428

```

429 <edm:Edmx xmlns:edm="http://docs.oasis-open.org/odata/ns/edm" Version="4.0">
430   <edm:Reference Uri=" http://redfish.dmtf.org/schemas/v1/ietf-inet-types.xml">
431     <edm:Include Namespace="ietf-inet-types.v1_0_0" Alias="inet" />
432   </edm:Reference>
433   . . .
434
435   <edm:DataServices>
436     <Schema Namespace="ietf_system" xmlns="urn.ietf.params.xml.ns.yang.ietf_system"
437       Alias="sys">
438       <Annotation Term="OData.LongDescription" String="[normative statement about RFC]/>
439
440       <EntityType Name="ietf_system" BaseType="Resource.v1_0_0.Resource">
441         <Annotation Term="Redfish.Yang.NodeType"
442           EnumMember ="Redfish.Yang.NodeTypes/module"/>
443         <Annotation Term="Redfish.Yang.contact" String="..." />
444         <Annotation Term="Redfish.Yang.description"
445           String="[text from description statement]" />
446         <Annotation Term="Redfish.Yang.revision" String="2014-12-18" />
447         <Annotation Term="OData.Description"
448           String="[text from description statement]"/>
449       </EntityType>
450     </Schema>
451
452     <Schema Namespace="ietf_system.v1_0_0" xmlns="urn.ietf.params.xml.ns.yang.ietf_system"
453       Alias="sys">
454       <EntityType Name="ietf_system" BaseType="ietf_dhcp.ietf_system">
455         <NavigationProperty Name="system" Type="system.system">
456           <Annotation Term="OData.Permissions" EnumMember="OData.Permission/ReadWrite"/>
457           <Annotation Term="OData.Description" String="" />
458           <Annotation Term="OData.LongDescription" String="" />
459           <Annotation Term="OData.AutoExpand"/>
460         </NavigationProperty>
461         <NavigationProperty Name="system_state" Type="system_state.system_state">
462           <Annotation Term="OData.Permissions" EnumMember="OData.Permission/Read"/>
463           <Annotation Term="OData.Description" String="" />
464           <Annotation Term="OData.LongDescription" String="" />
465           <Annotation Term="OData.AutoExpand"/>
466         </NavigationProperty>
467       </EntityType>
468     </Schema>
469   </edm:DataServices>
470 </edm:Edmx>

```

476 Table 3 shows the mapping of the *module* statement's sub-statements.

477

Table 3 – Module statement mapping

Statement	Mapping
anyxml	See clause 6.10
augment	See clause 6.15
choice	See clause 6.9
contact	<Annotation Term="Redfish.Yang.contact" String="[text from <i>contact</i> statement]"/>
container	See clause 6.4.4
description	See clause 6.25
deviation	See clause 6.21
extension	See clause 6.16.2
feature	See clause 6.19
grouping	See clause 6.11
identity	See clause 6.16
import	See clause 6.1.3
include	See clause 6.1.4
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
namespace	See clause 6.1.5
notification	See clause 6.1.3
organization	<Annotation Term="Redfish.Yang.organization" String="[text from <i>organization</i> statement]"/>
prefix	See clause 6.1.6.
reference	See clause 6.26
revision	<Annotation Term="Redfish.Yang.revision" String="[text from <i>revision</i> statement]"> <Annotation Term="Redfish.Yang.description" String="[text from <i>description</i> statement]"/> <Annotation Term="Redfish.Yang.reference" String="[text from <i>reference</i> statement]"/> </Annotation>
rpc	See clause 6.12.1
typedef	See clause 6.2.1
uses	See clause 6.12
yang-version	<Annotation Term="Redfish.Yang.yang_version" String="[Text from <i>yang-version</i> statement]"/>

478 6.1.3 Import statement

479 The *import* statement is mapped to a <edmx:Reference> tag. The *import* statement text is used to
 480 synthesize the value of the Uri and Namespace attributes. The *prefix* statement is mapped to value of the
 481 tag's Alias attribute.

482 Open the import target and read the YANG module's namespace to fill in the Namespace attribute of the
 483 Edmx:Include statement.

484 The YANG import statement is shown below.

```
485     prefix "dhcp";
486     import <import_value> {
487     prefix <prefix_value>;
488     }
```

489 The resultant Redfish CSDL is shown below.

```
490     <edmx:Reference Uri="<uri_value>">
491     <edmx:Include Namespace="<namespace_value>" Alias="<alias_value>" />
492     </edmx:Reference>
```

493 In which

- 494 • <uri_value> = http://redfish.dmtf.org/schemas/v1/<import_value>.xml
- 495 • <namespace_value> = <import_value>.v1_0_0
- 496 • <alias_value> = <prefix_value>

497 The YANG import statement from DHCP is shown below.

```
498     prefix "dhcp";
499     import ietf-inet-types {
500     prefix "inet";
501     }
```

502 The resultant Redfish CSDL is shown below.

```
503     <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/ietf-inet-types.xml">
504     <edmx:Include Namespace="ietf-inet-types.v1_0_0" Alias="inet" />
505     </edmx:Reference>
```

506 Table 4 shows the mapping of the *submodule* statement's sub-statements.

507 **Table 4 – Import statement mapping**

Statement	Mapping
prefix	<edmx:Include Alias="[text of prefix statement]"
revision-date	<Annotation Term="Redfish.Yang.revision_date" String="[text from revision-date statement]"/>

508 **6.1.4 Include statement**

509 From RFC6020, the "include" statement is used to make content from a submodule available to that
 510 submodule's parent module, or to another submodule of that parent module. The argument is an identifier
 511 that is the name of the submodule to include.

512 Modules are only allowed to include submodules that belong to that module, as defined by the "belongs-
 513 to" statement. Submodules are only allowed to include other submodules belonging to the same module.

514 Open the include target and read the YANG module's namespace to fill in the Namespace attribute of the
 515 Edmx:Include statement.

516 6.1.5 Namespace statement

517 The namespace statement is mapped to the OData <schema> tag.

518 The YANG *namespace* statement is shown below.

```
519 module <module value> {
520     namespace <namespace value>;
521     . . .
522 }
```

523 The resultant CSDL is shown below.

```
524 <schema Namespace="<Namespace value>" xmlns="<xmlns value>"
```

525 In which

- 526 • <Namespace value> = <module name>.v1_0_0
- 527 • <xmlns value> = <namespace value>"

528 The YANG code from DHCP is shown below.

```
529 module huawei-dhcp {
530     namespace "urn:ietf:params:xml:ns:yang:ietf-dhcp";
531     . . .
532 }
```

533 The resultant CSDL is shown below. In manual mapping, mapped to <schema xmlns value>

```
534 e.g. <schema Namespace="ietf_dhcp.v1_0_0" xmlns="urn:ietf:params:xml:ns:yang:ietf-dhcp">
```

535 6.1.6 Prefix statement

536 See clause 6.1.3, which also describes the *prefix* statement.

537 6.2 Submodule statement

538 While the primary unit in YANG is a module, a YANG module can itself be constructed out of several
 539 submodules. The "submodule" statement defines the submodule's name, and groups all statements that
 540 belong to the submodule together. The "submodule" statement's argument is the name of the submodule,
 541 followed by a block of sub-statements that hold detailed submodule information.

542 The YANG *submodule* is depicted as follows:

```
543 submodule: [submodule-name]
```

544 The resultant Redfish construct is a singleton resource. The modified-submodule-name is created by
 545 changing the dashes "-" to underscores "_" in the submodule-name.

```
546 ./NetworkDevices/{id}/[modified-name]
```

547 The following is example YANG code for a *submodule* statement.

```
548 submodule acme-types {
549     belongs-to "acme-system" {
550         prefix "acme";
551     }
552     import ietf-yang-types {
553         prefix "yang";
554     }
555     organization "ACME Inc.";
556     contact
557         "Joe L. User
558         ACME, Inc. . . . ";
```

559
560
561
562
563
564

```
description "This submodule defines common ACME types.";
revision "2007-06-09" {
    description "Initial revision.";
}
...
}
```

565 Table 5 shows the mapping of the *submodule* statement's sub-statements.

566 **Table 5 – Submodule statement mapping**

Statement	Mapping
anyxml	See clause 6.10
augment	See clause 6.15
belongs-to	See clause 6.2.1
choice	See clause 6.9
contact	<Annotation Term="Redfish.Yang.contact" String="[text from <i>contact</i> statement]"/>
container	See clause 6.4.4
description	See clause 6.25
deviation	See clause 6.21
extension	See clause 6.16.2
feature	See clause 6.19
grouping	See clause 6.11
identity	See clause 6.16
import	See clause 6.1.3
include	See clause 6.1.4
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
namespace	See clause 6.1.5
notification	See clause 6.1.3.
organization	<Annotation Term="Redfish.Yang.organization" String="[text from <i>organization</i> statement]"/>
reference	See clause 6.26
revision	<Annotation Term="Redfish.Yang.revision" String="[text from <i>revision</i> statement]"> <Annotation Term="Redfish.Yang.description" String="[text from <i>description</i> statement]"/> <Annotation Term="Redfish.Yang.reference" String="[text from <i>reference</i> statement]"/> </Annotation>
rpc	See clause 6.12.1
typedef	See clause 6.2.1
uses	See clause 6.12
yang-version	<Annotation Term="Redfish.Yang.yang_version" String="[Text from <i>version</i> statement]"/>

567 **6.2.1 Belongs-to statement**

568 From RFC602, the "belongs-to" statement specifies the module to which the submodule belongs. The
 569 argument is an identifier that is the name of the module. A submodule **MUST** only be included by the
 570 module to which it belongs, or by another submodule that belongs to that module.

571 The mandatory "prefix" substatement assigns a prefix for the module to which the submodule belongs.

572 The CSDL for the *belongs-to* statement is shown below.

```
573 <Annotation Term="Redfish.Yang.belongs to" String="[text from belongs-to statement]">
574 <Annotation Term="Redfish.Yang.prefix" String="[text from prefix statement]"/>
575 </Annotation>
```

576 **6.3 Typedef statement**

577 The "typedef" statement defines a new type that may be used locally in the module, in modules or
 578 submodules which include it, and by other modules that import from it. The new type is called the "derived
 579 type", and the type from which it was derived is called the "base type". All derived types can be traced
 580 back to a YANG built-in type.

581 There is no YANG depiction of a YANG *typedef* statement.

582 **6.3.1 Mapping YANG code to Redfish CSDL**

583 An example of the *typedef* statement from RFC 6991 (Common YANG data types) is shown below.

```
584 typedef gauge32 {
585     type uint32;
586     description "...";
587     reference "...";
588 }
```

589 The resultant Redfish construct is a TypeDefinition shown below.

```
590 <TypeDefinition Name="gauge32" UnderlyingType="Edm.Redfish.Yang.uint32">
591 <Annotation Term="Redfish.Yang.description" String="..."/>
592 <Annotation Term="Redfish.Yang.reference" String="..."/>
593 </TypeDefinition>
```

594 Another example of the *typedef* statement from RFC 6991 (Common YANG data types) is shown below.
 595 This one with a non-built-in type. Instead, *listen-ipv4-address* is derived from the exist type *inet:ipv4-*
 596 *address*.

```
597 typedef listen-ipv4-address {
598     type inet:ipv4-address;
599     default "0.0.0.0";
600 }
```

601 The resultant Redfish construct is a TypeDefinition declaration in the CSDL

```
602 <TypeDefinition Name="listen_ipv4_address" UnderlyingType="Edm.String">
603 <Annotation Term="Validation.Pattern" String="^[0-9]{1,3}\.\.]{3}[0-9]{1,3}$"/>
604 <Annotation Term="Redfish.Yang.default" String="0.0.0.0"/>
605 </TypeDefinition>
```

606 Table 6 shows the mapping of the *typedef* statement's sub-statements.

607 **Table 6 – Typedef statement mapping**

Statements	Mapping
default	See clause 6.3.2
description	See clause 6.25

Statements	Mapping
reference	See clause 6.26
status	See clause 6.24
type	UnderlyingType = <type_name>
units	<Annotation Term="Redfish.Yang.units" String="Text from units statement"/>

608 **6.3.2 Default statement**

609 The default value from the typedef statement is used, if the leaf or leaf-list statements does not have a
 610 default sub-statement present, use the default value from the typedef of the leaf or leaf-list type sub-
 611 statement to set the CSDL DefaultValue of the leaf or leaf-list corresponding property.

612 The default statement shall be mapped to an annotation in the CSDL and the value of the DefaultValue
 613 attribute of the Property property. The annotation shall be of the form shown below.

```
614 <Annotation Term="Redfish.Yang.default" String="Text from default statement"/>
```

615 The resultant Redfish CSDL for the example above is shown below.

```
616 <Property Name="listen_ipv4_address", Type="inet.ipv4_address",  

    617     DefaultValue="0.0.0.0" >  

    618     <Annotation Term="Redfish.Yang.YangType" String="inet.ipv4address"/>  

    619     <Annotation Term="Redfish.Yang.default" String="0.0.0.0"/>  

    620     . . .  

    621 </Property>
```

622 **6.4 Type statement**

623 From RFC6020, the "type" statement takes as an argument a string that is the name of a YANG built-in
 624 type or a derived type, followed by an optional block of sub-statements that are used to put further
 625 restrictions on the type. The restrictions that can be applied depend on the type being restricted.

626 Table 7 shows the list of YANG built-in types.

628 **Table 7 – Built in YANG types**

Name	Description	CSDL Mapping
binary	Any binary data	Edm.Binary
bits	A set of bits or flags	Edm.Binary
boolean	"true" or "false"	Edm.Boolean
date-and-time	Date and time	Edm.DateTimeOffset
decimal64	64-bit signed decimal number	Edm.Decimal
empty	A leaf that does not have any value	See clause 6.4.4.1
enumeration	Enumerated strings	See clause 6.4.4.2
identityref	A reference to an abstract identity	See clause 6.4.4.2
instance-identifier	References a data tree node	Redfish.Yang.instance_identifier
int8	A 8-bit signed integer	Edm.Sbyte
int16	A 16-bit signed integer	Edm.Int16

Name	Description	CSDL Mapping
int32	A 32-bit signed integer	Edm.Int32
int64	A 64-bit signed integer	Edm.Int64
leafref	A reference to a leaf reference	See clause 6.4.4.3
string	A human readable string	Edm.String
uint8	A 8-bit unsigned integer	Edm.Byte
uint16	A 16-bit unsigned integer	Redfish.Yang.uint16
uint32	A 32-bit unsigned integer	Redfish.Yang.uint32
uint64	A 64-bit unsigned integer	Redfish.Yang.uint64
union	A choice of member types	See clause 6.4.4.2

629 In Redfish.Yang.Types, there are TypeDefintion's that reflect the above table.

```
630 <TypeDefinition Name="uint16" UnderlyingType="Edm.Int32" />
631 <TypeDefinition Name="uint32" UnderlyingType="Edm.Int64" />
632 <TypeDefinition Name="uint64" UnderlyingType="Edm.Decimal" />
```

633 The type statement is mapped to following annotation

```
634 <Annotation Term="Redfish.Yang.YangType" String="[value of type statement]"/>
```

635 There is no YANG depiction of a YANG *type* statement.

636 6.4.1 Mapping YANG code to Redfish CSDL

637 A *type* statement from DHCP is shown below.

```
638 leaf enable {
639     description "Enable or disable dhcp relay function";
640     type "boolean";
641     default "false";
642     config "true";
643 }
```

644 The resultant Redfish is shown below. The value of the type statement is mapped to the Type value in the
645 Property definition. The annotation is also added to preserve the original YANG type.

```
646 <Property Name="enable", Type="edm:Boolean" >
647     <Annotation Term="Redfish.Yang.YangType" String="boolean"/>
648     . . .
649 </Property>
```

650 Table 8 shows the mapping of the *type* statement's sub-statements.

651 **Table 8 – Type statement mapping**

Statements	Mapping
base	<Annotation Term="Redfish.Yang.base" String="the_yang_statement_base string"/>
bit	<Annotation Term="Redfish.Yang.bit" String="bit_name"/> <Annotation Term="Redfish.Yang.position" Redfish.Yang.uint32=bit_position/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> <Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus"/> </Annotation>

Statements	Mapping
enum	<p>Instance of edm.Member where Name = "enum_name"</p> <pre><Annotation> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> <Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus"/ </Annotation></pre>
length	<pre><Annotation Term="Redfish.Yang.length" String="the length sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="Text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="Text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> </Annotation></pre>
path	See clause 6.4.2
pattern	<pre><Annotation Term="Redfish.Yang.pattern" String="[text from the pattern statement]"> <Annotation Term="Redfish.Yang.error_message" String="[text from error-message statement]" /> <Annotation Term="Redfish.Yang.error_app_tag" String="[text from error-app-tag statement]" /> <Annotation Term="Redfish.Yang.description" String="[text from description statement]" /> <Annotation Term="Redfish.Yang.reference" String="[text from reference statement]" /> </Annotation></pre>
range	<pre><Annotation Term="Redfish.Yang.range" String="the range sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="[text from error-message statement]" /> <Annotation Term="Redfish.Yang.error_app_tag" String="[text from error-app-tag statement]" /> <Annotation Term="Redfish.Yang.description" String="[text from description statement]" /> <Annotation Term="Redfish.Yang.reference" String="[text from reference statement]" /> </Annotation></pre>
required-instance	See clause 6.4.3
type	Ignore. The <i>type</i> sub-statement is not supported.

652 **6.4.2 Path statement**

653 The "path" statement, takes as an argument a string that MUST refer to a leaf or leaf-list node. The
 654 syntax for a path argument is a subset of the XPath abbreviated syntax. Predicates are used only for
 655 constraining the values for the key nodes for list entries. Each predicate consists of exactly one equality
 656 test per key, and multiple adjacent predicates MAY be present if a list has multiple keys.

657 **6.4.3 require-instance statement**

658 The "require-instance" statement MAY be present if the type is "instance-identifier". It takes as an
 659 argument the string "true" or "false".

660 If "require-instance" is "true", it means that the instance being referred MUST exist for the data to be valid.

661 If "require-instance" is "false", it means that the instance being referred MAY exist in valid data.

662 The CSDL annotation is show below.

```
663 <Annotation Term="Redfish.Yang.require_instance"
664     String="[text from require-instance statement]"/>
```

665 6.4.4 Mapping special types

666 Returning to Table 7, some of the built-in YANG types are mapped to something more complex than a
667 simple annotation. The following clause specifies that mapping of each of these special built-in YANG
668 types.

669 6.4.4.1 Empty type

670 From RFC6020, the empty built-in type represents a leaf that does not have any value, it conveys
671 information by its presence or absence.

672 Neither CSDL nor json-schema support this semantic.

673 The *empty* statement is mapped to a read-only string that only returns the name of the leaf.

674 The *empty* statement shall be mapped to an annotation in the CSDL and a Property that only contains the
675 value of the empty statement.

676 The YANG depiction is shown below.

```
677 +--ro is-router?          empty
```

678 The resultant Redfish CSDL for the example above is shown below.

```
679 <Property Name="is router", Type="Redfish.Yang.empty", DefaultValue="is router" >
680   <Annotation Term="Redfish.Yang.YangType" String="empty"/>
681 </Property>
```

682 6.4.4.2 Enumeration type

683 From RFC 6020, the enumeration built-in type represents values from a set of assigned names.

684 The enumeration type will be mapped to Odata EnumType.

685 The YANG code for the enumeration type from RFC 6991 (Common YANG Types) is shown below.

```
686 typedef ip-version {
687     type enumeration {
688         enum unknown {
689             value "0";
690             description
691                 "An unknown or unspecified version of the Internet
692                 protocol.";
693         }
694         enum ipv4 {
695             value "1";
696             description
697                 "The IPv4 protocol as defined in RFC 791.";
698         }
699         enum ipv6 {
700             value "2";
701             description
702                 "The IPv6 protocol as defined in RFC 2460.";
703         }
704     }
705 }
```

706 The resultant Redfish CSDL is shown below. (system example)

```
707 <EnumType Name="association_typeEnumeration">
708   <Member Name="server">
```

```

709     <Annotation Term="Redfish.Yang.enum" String="server"/>
710     <Annotation Term="OData.Description"
711       String="Use client association mode.[...]" />
712   </Member>
713   <Member Name="peer">
714     <Annotation Term="Redfish.Yang.enum" String="peer"/>
715     <Annotation Term="OData.Description"
716       String="Use symmetric active association mode.[...]" />
717   </Member>
718   <Member Name="pool">
719     <Annotation Term="Redfish.Yang.enum" String="pool"/>
720     <Annotation Term="OData.Description"
721       String="Use client association mode with one or more of the NTP
722 servers.[...]" />
723   </Member>
724 </EnumType>
725 Identifyref Type

```

726 From RFC6020, the identityref type is used to reference an existing identity.

727 The "base" statement, which is a substatement to the "type" statement, MUST be present if the type is
728 "identityref". The argument is the name of an identity, as defined by an "identity" statement.

729 The YANG code from RFC7223

```

730     leaf type {
731       type identityref {
732         base interface-type;
733       }
734       mandatory true;
735       description "...";
736       reference
737         "RFC 2863: The Interfaces Group MIB - ifType";
738     }

```

739 6.4.4.3 Leafref type

740 From RFC6020, the leafref built-in type is used to reference a particular leaf instance in the data tree. The
741 "path" sub-statement selects a set of leaf instances, and the leafref value space is the set of values of
742 these leaf instances. The "path" statement MUST be present if the type is "leafref".

743 The value of Leaftype is set to the type of the Edm.Property for the leaf is the type of the leafref's target
744 leaf node. Returns the value of another leaf.

745 The YANG code from RFC7223

```

746     typedef interface-state-ref {
747       type leafref {
748         path "/if:interfaces-state/if:interface/if:name";
749       }
750       description
751         "This type is used by data models that need to reference
752         the operationally present interfaces.";
753     }
754
755     leaf-list higher-layer-if {
756       type interface-state-ref;
757       description
758         "A list of references to interfaces layered on top of this
759         interface.";
760       reference
761         "RFC 2863: The Interfaces Group MIB - ifStackTable";
762     }
763
764
765

```

766 The resultant CSDL is shown below (path value is considered a opaque string, therefore the colons
767 remain.

```
768 <Property Name="<name of the leaf with has the type specified by the leafref",
769 Type="(derived by derferencing the path and using the type of dereferenced target" >
770 <Annotation Term="Redfish.Yang.YangType" String="leafref">
771 <Annotation Term="Redfish.Yang.path" String="if:interfaces/if:interface/if:name
772 "/>
773 </Annotation>
774 </Property>
775
```

776 An example

```
777 <Property Name="higher-layer-if", Type="string" >
778 <Annotation Term="Redfish.Yang.YangType" String="leafref">
779 <Annotation Term="Redfish.Yang.path" String="if:interfaces/if:interface/if:name
780 "/>
781 </Annotation>
782 </Property>
783
```

784 6.4.4.4 Union type

785 From RFC6020, the union built-in type represents a value that corresponds to one of its member types.

786 A member type can be of any built-in or derived type, except it MUST NOT be one of the built-in types
787 "empty" or "leafref".

788 For example:

```
789 type union {
790     type int32;
791     type enumeration {
792         enum "unbounded";
793     }
794 }
```

795 6.4.4.4.1 Mockup

796 The JSON payload would include an @odata.type annotation to specify the type of the actual IPAddress:

```
797 {
798     ...
799     "IPAddress": "...",
800     "IPAddress@odata.type": "#IP.IPV4_no_zone"
801 }
```

802 6.4.4.4.2 Mapping YANG code to Redfish CSDL

803 The union statement can be mapped two ways in CSDL.

804 One option is that the IPAddress property can be annotated with a Redfish.Yang.Union annotation, which
805 specifies the possible values within a collection.

```
806 <Property Name="IPAddress" Type="Edm.Primitive">
807 <Annotation Term="Redfish.Yang.Union">
808 <Collection>
809 <String>"IPV4_no_zone"</String>
810 <String>"IPV6_no_zone"</String>
811 </Collection>
812 </Annotation>
813 </Property>
```

814 The Redfish.Yang.Union annotation is specified elsewhere, as a collection type.

```
815 <Term Name="Union" Type="Collection(String)">
816 <Annotation Term="OData.Description" String=""/>
817 </Term>
```

818 Another options is that the IPAddress property specifies a property type definition for the union.

```
819 <Property Name="IPAddress" Type="IP.ip_no_zone"/>
```

820 The type definition declares that ip_no_zone has an underlying type of "Edm.Primitive and specifies the possible types.

```
822 <TypeDefinition Name="ip_no_zone" UnderlyingType="Edm.Primitive">
823 <Annotation Term="Redfish.Yang.Union">
824 <Collection>
825 <String>"IP.IPV4 no zone"</String>
826 <String>"IP.IPV6_no_zone"</String>
827 </Collection>
828 </Annotation>
829 </TypeDefinition>
```

830 6.5 Container statement

831 From RFC6020, the "container" statement is used to define an interior data node in the schema tree. It
832 takes one argument, which is an identifier, followed by a block of sub-statements that holds detailed
833 container information.

834 A container node does not have a value, but it has a list of child nodes in the data tree. The child nodes
835 are defined in the container's sub-statements.

836 YANG supports two styles of containers, those that exist only for organizing the hierarchy of data nodes,
837 and those whose presence in the configuration has an explicit meaning.

838 6.5.1 Mapping the YANG depiction to Redfish mockup

839 The YANG *container* is depicted is show below.

```
840 +--[container-name]
841
842 +--relay (DHCP example)
843 +--rw dhcpRelayIfCfags
844 +--rw dhcpRelayServerGroups
845 +--r dhcpRelayStatistics
```

846 The resultant Redfish construct is a singleton resource

```
847 ./NetworkDevices/{id}/[module-name]/[container-name]
848
849 ./NetworkDevices/{id}/ietf_dhcp/relay (DHCP example)
```

850 A mockup of the "relay" resource is shown below. It contains navigation links for the containers contained
851 by "relay".

```
852 {
853   "@Redfish.Copyright": "",
854   "@odata.context":
855   "/redfish/v1/$metadata#NetworkDevices/Member/ietf_dhcp/relay/$entity",
856   "@odata.type": "#relay.1.0.0.relay",
857   "@odata.id": "/redfish/v1/NetworkDevices/SW 15/ietf_dhcp/relay",
858
859   "Id": "relay",
860   "Name": "DHCP Relay Service",
861 }
```

```

862     "dhcpRelayIfCfgs": {
863         "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/relay/dhcpRelayIfCfgs"
864     },
865     "dhcpRelayServerGroups": {
866         "@odata.id":
867         "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/relay/dhcpRelayServerGroups"
868     },
869     "dhcpRelayStatistics": {
870         "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/relay/dhcpRelayStatistics"
871     }
872 }

```

873 6.5.2 Mapping YANG code to Redfish CSDL

874 The YANG code for the "relay" container statement from DHCP is shown below.

```

875 container relay {
876     container dhcpRelayIfCfgs {
877         . . .
878     }
879     Container dhcpRelayServerGroups {
880         . . .
881     }
882     Container dhcpRelayStatistics {
883         . . .
884     }
885 }

```

886 The resultant CSDL fragment for relay container statement is shown below. There is a Navigation
887 property for each sub-container.

```

888 <?xml version="1.0" encoding="UTF-8"?>
889 <!-- Copyright 2014-2015 Distributed Management Task Force, Inc. (DMTF). All rights
890 reserved.-->
891 <edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
892
893     <edmx:Reference Uri="http://docs.oasis-
894 open.org/odata/odata/v4.0/cs01/vocabularies/Org.OData.Core.V1.xml">
895     <edmx:Include Namespace="Org.OData.Core.V1" Alias="OData" />
896 </edmx:Reference>
897 <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/Resource.xml">
898     <edmx:Include Namespace="Resource.1.0.0" />
899 </edmx:Reference>
900
901 <edmx:DataServices>
902
903     <Schema Namespace="relay" xmlns="http://docs.oasis-open.org/odata/ns/edm">
904     <EntityType Name="relay" BaseType="Resource.1.0.0.Resource">
905         <Annotation Term="OData.Description" String=""/>
906         <Annotation Term="OData.AdditionalProperties" Bool="false"/>
907     </EntityType>
908 </Schema>
909
910     <Schema Namespace="relay.1.0.0" xmlns="http://docs.oasis-open.org/odata/ns/edm">
911     <EntityType Name="relay" BaseType="relay.relay">
912         Annotation Term="Redfish.Yang.NodeType"
913         EnumMember ="Redfish.Yand.NodeTypes/container"/>
914     <Annotation Term="OData.Description" String=""/>
915     <Annotation Term="OData.AdditionalProperties" Bool="false"/>
916     <NavigationProperty Name="dhcpRelayIfCfgs"
917         Type="dhcpRelayIfCfgsCollection.dhcpRelayIfCfgsCollection"
918         ContainsTarget="true">
919         <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
920         <Annotation Term="OData.Description" String=""/>
921         <Annotation Term="OData.LongDescription" String=""/>
922         Annotation Term="OData.AutoExpandReferences"/>
923     </NavigationProperty>
924     <NavigationProperty Name="dhcpRelayServerGroups"
925         Type="dhcpRelayServerGroupsCollection.dhcpRelayServerGroupsCollection"

```

```

926         ContainsTarget="true">
927         <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
928         <Annotation Term="OData.Description" String=""/>
929         <Annotation Term="OData.LongDescription" String=""/>
930         <Annotation Term="OData.AutoExpandReferences"/>
931     </NavigationProperty>
932     <NavigationProperty Name="dhcpRelayStatistics"
933         Type="dhcpRelayStatistics.dhcpRelayStatistics"
934         ContainsTarget="true">
935         <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
936         <Annotation Term="OData.Description" String=""/>
937         <Annotation Term="OData.LongDescription" String=""/>
938         <Annotation Term="OData.AutoExpandReferences"/>
939     </NavigationProperty>
940 </EntityType>
941 </Schema>
942
943 </edmx:DataServices>
944 </edmx:Edmx>
    
```

945 Table 9 shows the mapping of the *container* statement's sub-statements.

946 **Table 9 – Container statement mapping**

Statement	Mapping
container	Recursion. See this clause.
list	See clause 6.8
leaf	See clause 6.6
leaf-list	See clause 6.7
presence	<Annotation Term="Redfish.Yang.presence" String="text from presence statement"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="Text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="Text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> </Annotation>
when	See clause 6.27
config	See clause 6.22
if-feature	See clause 6.20
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1
choice	See clause 6.9
grouping	See clause 6.11
uses	See clause 6.12
anyxml	See clause 6.10

947 **6.6 Leaf statement**

948 From RFC6020, the "leaf" statement is used to define a leaf node in the schema tree. It takes one
 949 argument, which is an identifier, followed by a block of sub-statements that holds detailed leaf
 950 information.

951 The *leaf* statement is mapped to a JSON property.

952 **6.6.1 Mapping YANG depiction to Redfish mockup**

953 The YANG depiction of the *leaf* statement is shown below.

```
954 +--[permission] [leaf-name] [leaf-type]
955
956 +--rw serverGroupName string (DHCP example)
```

957 The resultant Redfish is a JSON property within a resource mockup.

```
958 [leaf-name]: "[value]"
959
960 "serverGroupName": "webservers" (DHCP example)
```

961 **6.6.2 Mapping YANG code to Redfish CSDL**

962 The YANG code for a *leaf* statement is shown below.

```
963 leaf clientRequestCount {
964     description "Client Request Count";
965     type uint32;
966     config "false";
967 }
```

968 The resultant CSDL fragment for the JSON properties is shown below.

```
969 <Property Name="clientRequestCount" Type="Redfish.Yang.uint32">
970     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
971     <Annotation Term="Redfish.Yang.YangType" String="uint32"/>
972     . . .
973 </Property>
```

974 Table 10 shows the mapping of the *leaf* statement's sub-statements.

975 **Table 10 – Leaf statement mapping**

Statement	Mapping
type	See clause 6.3.2
units	<Annotation Term="Redfish.Yang.units" String="[text from units statement]"/>
default	See clause 6.3.2
mandatory	One of <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/false"/> <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/true"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="Text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="Text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> </Annotation>

Statement	Mapping
config	See clause 6.22
if-feature	See clause 6.20
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

976 6.7 Leaf-list statement

977 The "leaf-list" statement is used to define an array of a particular type. The "leaf-list" statement takes one
 978 argument, which is an identifier, followed by a block of sub-statements that holds detailed leaf-list
 979 information.

980 The leaf-list statement is mapped to JSON property array which the mockup.

981 6.7.1 Mapping YANG depiction to Redfish mockup

982 The YANG *leaf-list* statement is depicted is shown below. The depiction is identical to the depiction of a
 983 *leaf* statement. One needs to consult the YANG code to view the statement.

```
984 +--[permission] [leaf-list-name] [leaf-type]
985
986 +--rw serverAddress inet:ipv4-address (DHCP example)
```

987 The resultant Redfish construct is a JSON array property within the resource mockup.

```
988 "[leaf-list-name]": [
989     "[value 1]",
990     "[value 2]"
991     . . .
992 ]
993
994 "serverAddress": [ (DHCP example)
995     "[ip address 1]",
996     "[ip address 2]"
997 ]
```

998 6.7.2 Mapping YANG code to Redfish CSDL

999 The YANG code from *leaf-list* statement of DHCP is shown below. (with

```
1000 leaf-list serverAddress {
1001     description "DHCP relay destination server IP address";
1002     type inet:ipv4-address;
1003     config "true";
1004 }
```

1005 The resultant CSDL fragment for the JSON properties is shown below.

```
1006 <Property Name="serverAddress" Type="Collection(Yang.inet:ipv4-address)">
1007     <Annotation Term="Redfish.Yang.NodeType"
1008         EnumMember="Redfish.Yang.NodeTypes/leaf_list" />
1009     <Annotation Term="Redfish.Yang.YangType" String="inet:ipv4-address" />
1010 </Property>
```


1011

Table 11 – Leaf-list statement mapping

Statement	Mapping
type	See clause 6.5
units	<Annotation Term="Redfish.Yang.units" String="Text from units statement"/>
max-elements	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=max_elements/>/true"/>
min-elements	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=min_elements/>/true"/>
ordered-by	<Annotation Term="Redfish.Yang.ordered_by" EnumMember="Redfish.Yang.ConfigPermission/false"/> <Annotation Term="Redfish.Yang.ordered_by" EnumMember="Redfish.Yang.ConfigPermission/true"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="Text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="Text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="Text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="Text from reference statement"/> </Annotation>
config	See clause 6.22
If-feature	See clause 6.20
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

1012 **6.8 List statement**

1013 A list defines a sequence of list entries. Each entry is like a structure or a record instance, and is uniquely
1014 identified by the values of its key leafs. A list can define multiple key leafs and may contain any number
1015 of child nodes of any type (including leafs, lists, containers, etc.)

1016 The *list* statement is mapped to a Redfish collection resource and its member resource.

1017 **6.8.1 Mapping YANG depiction to Redfish mockup**

1018 The YANG depiction of the *list* statement is shown below.

1019
1020
1021
1022
1023
1024
1025
1026
1027
1028

```

+--rw [list-name]
| +--rw [list-member-name]* [[name-of-member]]
| +--rw [name-of-member] string
. . .
+--rw dhcpRelayIfCfgs (DHCP example)
| +--rw dhcpRelayIfCfg* [ifName]
| +--rw ifName string
. . .
    
```

1029 The resultant Redfish is a collection resource and singleton resources. The value of the "ifName" leaf
1030 statement is used at the name of the member of the collection.

```
1031 ./NetworkDevices/{id}/ietf_dhcp/relay/dhcpRelayIfCfgs
1032 ./NetworkDevices/{id}/ietf_dhcp/relay/dhcpRelayIfCfgs/[Text of ifName leaf statement]
```

1033 A mockup of the "dhcpRelayIfCfgs" collection resource is shown below.

```
1034 {
1035     . . .
1036     "@odata.id": "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/relay/dhcpRelayIfCfgs",
1037
1038     "Name": "Collection of interface configurations for DHCP relay service",
1039     "Members@odata.count": 1,
1040     "Members": [
1041         {"@odata.id":
1042          "/redfish/v1/NetworkDevices/SW_15/ietf_dhcp/relay/dhcpRelayIfCfgs/IF_foo"
1043         }
1044     ]
1045 }
```

1046 A mockup of the 'IF_foo' singleton dhcpRelayIfCfg resource is show below.

```
1047 {
1048     . . .
1049     "Id": "IF_foo",
1050     "Name": "Interface configuration for a DHCP relay service",
1051
1052     "ifName": "IF foo",
1053     "enable": "TRUE",
1054     "serverAddress": "192.168.1.10",
1055     . . .
1056 }
1057 }
```

1058 6.8.2 Mapping YANG code to Redfish CSDL

1059 The YANG code for the *list* statement from the DHCP is shown below.

```
1060 list dhcpRelayIfCfg {
1061     key "ifName";
1062     leaf ifName {
1063         description "Specify the interface name that dhcp relay configured on";
1064         type "string";
1065         config "true";
1066     }
1067     . . .
1068 }
1069 }
```

1070 The CSDL for the collection resource is shown below.

```
1071 <edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
1072     . . .
1073     <edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/namespace="dhcpRelayIfCfg.xml">
1074         <edmx:Include Namespace="dhcpRelayIfCfg"/>
1075     </edmx:Reference>
1076     <edmx:DataServices>
1077         <Schema Namespace="Namespace="dhcpRelayIfCfgsCollection"
1078             xmlns="http://docs.oasis-open.org/odata/ns/edm" >
1079             <EntityType Name="dhcpRelayIfCfgsCollection"
1080                 BaseType="Resource.1.0.0.ResourceCollection">
1081                 <NavigationProperty Name="Members"
1082                     Type="Collection(dhcpRelayIfCfg.dhcpRelayIfCfg)" >
```

1086
1087
1088
1089
1090
1091
1092
1093
1094

```

        <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
        <Annotation Term="OData.Description" String=""/>
        <Annotation Term="OData.AutoExpandReferences"/>
    </NavigationProperty>
</EntityType>
</Schema>

</edmx:DataServices>
</edmx:Edmx>
    
```

1095 Table 12 shows the mapping of the *list* statement.

1096

Table 12 – List statement mapping

Statement	Mapping
container	See clause 6.4.4
list	See clause 6.8
leaf	See clause 6.6
leaf-list	See clause 6.7
key	See clause 6.8.3
max-elements	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=max_elements/>/true"/>
min-elements	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=min_elements/>/true"/>
ordered-by	One of <Annotation Term="Redfish.Yang.ordered_by" EnumMember="Redfish.Yang.ConfigPermission/false"/> <Annotation Term="Redfish.Yang.ordered_by" EnumMember="Redfish.Yang.ConfigPermission/true"/>
must	<Annotation Term="Redfish.Yang.must" String="text from the XPath statement"> <Annotation Term="Redfish.Yang.error_message" String="text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="text from reference statement"/> </Annotation>
config	See clause 6.22
if-feature	See clause 6.20
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1
choice	See clause 6.9
grouping	See clause 6.11
uses	See clause 6.12
anyxml	See clause 6.10
unique	<Annotation Term="Redfish.Yang.unique" String="text from unique statement"/>
when	See clause 6.27

1097 6.8.3 Key statement

1098 From RFC6020, the "key" statement, which MUST be present if the list represents configuration, and
 1099 MAY be present otherwise, takes as an argument a string that specifies a space-separated list of leaf
 1100 identifiers of this list. A leaf identifier MUST NOT appear more than once in the key. Each such leaf
 1101 identifier MUST refer to a child leaf of the list. The leaves can be defined directly in sub-statements to the
 1102 list, or in groupings used in the list.

1103 The combined values of all the leafs specified in the key are used to uniquely identify a list entry.

1104 The *key* statement value is a space separated sting of leaf names. Typically there will be only one key
 1105 token in the key string but there are a couple cases of 2 or more keys.

1106 The following is a *key* sub-statement from DHCP

```
1107 <EntityType Name="dhcpRelayIfCfgs">
1108   <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/list"/>
1109   . . .
1110   <Annotation Term="Redfish.Yang.key" String=" the yang key string"/>
1111   <Key>
1112     <PropertyRef Name="ifName" />
1113   </Key>
1114   . . .
1115 </EntityType>
```

1116 The set of names constitutes the key for this list. The *ithKeyName* corresponds to the *ith* string token in
 1117 the key string. We add annotations containing the original yang information in addition to the actual
 1118 translation to make understanding the translated schema clearer.

```
1119 <EntityType Name="theListName">
1120   <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/list"/>
1121   . . .
1122   <Annotation Term="Redfish.Yang.key" String=" the yang key string"/>
1123   <Key>
1124     <PropertyRef Name="firstKeyName" />
1125     ...
1126     <PropertyRef Name="ithKeyName" />
1127     ...
1128     <PropertyRef Name="lastKeyName" />
1129   </Key>
1130   . . .
1131   <Property Name="firstLeafName" Type="translatedLeafType">
1132     <Annotation Term="Redfish.Yang.NodeType" EnumMember
1133     ="Redfish.Yang.NodeTypes/leaf"/>
1134     <Annotation Term="Redfish.Yang.YangType" String="theOriginalYangType"/>
1135   </Property>
1136   <Property Name="nthLeafName" Type="translatedLeafType">
1137     <Annotation Term="Redfish.Yang.NodeType" EnumMember
1138     ="Redfish.Yang.NodeTypes/leaf"/>
1139     <Annotation Term="Redfish.Yang.YangType" String="theOriginalYangType"/>
1140   </Property>
1141   <Property Name="lastLeafName" Type="translatedLeafType">
1142     <Annotation Term="Redfish.Yang.NodeType" EnumMember
1143     ="Redfish.Yang.NodeTypes/leaf"/>
1144     <Annotation Term="Redfish.Yang.YangType" String="theOriginalYangType"/>
1145   </Property>
1146   . . .
1147 </EntityType>
```

1148 6.9 Choice statement

1149 From RFC6020, the "choice" statement defines a set of alternatives, only one of which may exist at any
 1150 one time. The argument is an identifier, followed by a block of sub-statements that holds detailed choice

1151 information. The identifier is used to identify the choice node in the schema tree. A choice node does not
1152 exist in the data tree.

1153 A choice consists of a number of branches, defined with the "case" sub-statement. Each branch contains
1154 a number of child nodes. The nodes from at most one of the choice's branches exist at the same time.).

1155 The choice statement maps to a Redfish collection resource and the key maps to members of the
1156 collection

1157 The choice statement maps to annotations for the choice and each of the cases with the containing
1158 resource. The positioning shall correspond to the position of the element within the case statement.

1159 The choice annotation will have as children the translated annotations for the directly dependent non-
1160 node YANG statements of the choice plus case statements of the choice plus a
1161 <Redfish.Yang.NodeName < Redfish.Yang.NodeType > />statement hierarchy for each node.

1162 Put all node elements from all cases directly in the parent node and annotate each one individually with a
1163 "choice annotation and a case annotation" s.

1164 Create Annotations to represent the choice/case structure. Create an Annotation for the choice itself in
1165 the context of its parent container. The choice annotation will have as children the translated annotations
1166 for the directly dependent non-node yang statements of the choice plus case statements of the choice
1167 plus a <Redfish.Yang.NodeName < Redfish.Yang.NodeType > />statement hierarchy for each node

1168 The nodes themselves will otherwise be translated in the context of the parent node of the choice
1169 statement as direct properties of the parent node plus EntityType objects as needed to translate list and
1170 container.

1171 6.9.1 Mapping the YANG depiction to Redfish mockup

1172 The YANG *choice* depiction from RFC7317 is shown below. For "timezone", there is choice between
1173 "timezone-name" and "timezone-utc-offset".

```
1174     +--ro (timezone)?
1175         +--:(timezone-name)
1176         | +--ro timezone-name? string
1177         +--:(timezone-utc-offset)
1178         +--ro timezone-utc-offset? int16
```

1179 The possible resultant Redfish mockups are the shown below.

```
1180     {
1181         "timezone-name": "Europe/Stockholm",
1182     }
1183
```

```
1184     {
1185         "timezone-utc-offset": "3",
1186     }
```

1187 6.9.2 Translating the YANG depiction to Redfish mockup

1188 The YANG code from RFC7317 is shown below.

```
1189     container clock {
1190         description "Configuration of the system date and time properties.";
1191
1192         choice timezone {
1193             description "The system time zone information.";
1194
1195             case timezone-name {
1196                 if-feature timezone-name;
1197                 leaf timezone-name {
1198                     type timezone-name;
1199                 }
1200             }
1201         }
1202     }
```

```

1199         description "The TZ database name to use for the system, such as
1200             'Europe/Stockholm'. ";
1201     }
1202 }
1203 case timezone-utc-offset {
1204     leaf timezone-utc-offset {
1205         type int16 {
1206             range "-1500 .. 1500";
1207         }
1208         units "minutes";
1209         description "The number of minutes to add to UTC ...";
1210     }
1211 }
1212 }
1213 }

```

1214 The resultant CSDL fragment for the DHCP service is shown below.

```

1215 <EntityType Name = "clock" >
1216     <Annotation Term="Redfish.Yang.description" String="Configuration of the system date
1217         and time properties."/>
1218     <Annotation Term="Redfish.Yang.choice" String="timezone">
1219         <Annotation Term="Redfish.Yang.description" String="The system time zone
1220 information."/>
1221         <Annotation Term="Redfish.Yang.case" String="timezone-name">
1222             <Annotation Term="Redfish.Yang.if feature" String="timezone-name"/>
1223             <Annotation Term="Redfish.Yang.NodeName" String="timezone-name" >
1224                 <Annotation Term="Redfish.Yang.NodeType"
1225                     EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
1226             </Annotation>
1227         </Annotation>
1228         <Annotation Term="Redfish.Yang.case" String="timezone-utc-offset">
1229             <Annotation Term="Redfish.Yang.NodeName" String="timezone-utc-offset" >
1230                 <Annotation Term="Redfish.Yang.NodeType"
1231                     EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
1232             </Annotation>
1233         </Annotation>
1234     </Annotation>
1235
1236     <Property Name = "timezone_name" Type = "timezone_name">
1237         <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
1238         <Annotation Term="Redfish.Yang.YangType" String="timezone-name"/>
1239         <Annotation Term="Redfish.Yang.description" String="The TZ database name to..."/>
1240         <Annotation Term="Redfish.Yang.choice" String="timezone"/>
1241         <Annotation Term="Redfish.Yang.case" String="timezone-name"/>
1242     </Property>
1243
1244     <Property Name = "timezone_utc_offset" Type = "int16">
1245         <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
1246         <Annotation Term="Redfish.Yang.YangType" String="int16">
1247             <Annotation Term="Redfish.Yang.range" String="-1500 .. 1500"/>
1248         </Annotation>
1249         <Annotation Term="Redfish.Yang.units" String=" minutes "/>
1250         <Annotation Term="Redfish.Yang.description"
1251             String="The number of minutes to add to UTC time to..."/>
1252         <Annotation Term="Redfish.Yang.choice" String="timezone"/>
1253         <Annotation Term="Redfish.Yang.case" String="timezone-utc-offset"/>
1254     </Property>
1255 </EntityType>
1256

```

1257 Table 13 shows the mapping of the *choice* statement's sub-statements.

1258

Table 13 – Choice statement mapping

Statements	Mapping
anyxml	See clause 6.10
case	See clause 6.9.3
choice	See clause 6.9
container	See clause 6.4.4
default	<Annotation Term="Redfish.Yang.default" String="the_yang_default_string"/>
description	See clause 6.25
if-feature	See clause 6.20
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
mandatory	One of <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/false"/> <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/true"/>
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

1259 6.9.3 Case

1260 From RFC6020, the "case" statement is used to define branches of the choice. It takes as an argument
1261 an identifier, followed by a block of sub-statements that holds detailed case information.

1262 The identifier is used to identify the case node in the schema tree. A case node does not exist in the data
1263 tree.

1264 See clause 6.9 for the mapping details.

1265 6.10 Anyxml statement

1266 From RFC6020, The "anyxml" statement defines an interior node in the schema tree. It takes one
1267 argument, which is an identifier, followed by a block of sub-statements that holds detailed anyxml
1268 information.

1269 The "anyxml" statement is used to represent an unknown chunk of XML. No restrictions are placed on the
1270 XML.

1271 The *anyxml* statement is mapped to an annotation within its parent container and parent annotation.

1272 6.10.1 Mapping YANG depiction to Redfish mockup

1273 An example of a YANG depiction of the *anyxml* statement has not been found.

1274 **6.10.2 Mapping YANG code to Redfish CSDL**

1275 The YANG code from *anyxml* statement from RFC6020 is shown below.

1276 `anyxml data;`

1277 The resultant CSDL is shown below.

```

1278 <Term Name="IsXml" Type="Edm.Boolean" Default="True">
1279   <Annotation Term="OData.Description" String="The string type contains XML"/>
1280 </Term>
1281
1282 <TypeDefinition Name="XmlBlock" UnderlyingType="Edm.String">
1283   <Annotation Term="Redfish.Yang.IsXml"/>
1284 </TypeDefinition>
1285
1286 <Property Name="myProperty" Type="Redfish.Yang.XmlBlock">
1287   [text from anyxml statement]
1288 </Property>
    
```

1289 Where "myProperty" is a unique name synthesized by appending a number to the string "Anyxml_".

1290 Table 14 shows the mapping of the *anyxml* statement's sub-statements.

1291 **Table 14 – Anyxml statement mapping**

Statement	Mapping
config	See clause 6.22
description	See clause 6.25
if-feature	See clause 6.20
mandatory	One of <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/false"/> <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/true"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="text from reference statement"/> </Annotation>
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

1292 **6.11 Grouping statement**

1293 From RFC6020, the "grouping" statement is used to define a reusable block of nodes, which may be used
 1294 locally in the module, in modules that include it, and by other modules that import from it. It takes one
 1295 argument, which is an identifier, followed by a block of sub-statements that holds detailed grouping
 1296 information.

1297 The "grouping" statement is not a data definition statement and, as such, does not define any nodes in
 1298 the schema tree. A grouping is like a "structure" or a "record" in conventional programming languages.

1299 The *grouping* and *uses* statement should be handled and resolved prior to mapping the YANG to CSDL.
 1300 Since the *grouping* statement does not define a node in the schema tree, there is no YANG depiction.

1301 The YANG code for the grouping statement from inet-types is shown below.

```

1302 import ietf-inet-types {
1303     prefix "inet";
1304 }
1305
1306     grouping endpoint {
1307         description "A reusable endpoint group.";
1308         leaf ip {
1309             type inet:ip-address;
1310         }
1311         leaf port {
1312             type inet:port-number;
1313         }
1314     }
    
```

1315 Table 15 shows the mapping of the *grouping* statement's sub-statements.

1316 **Table 15 – Grouping statement mapping**

Statement	Mapping
choice	See clause 6.9
container	See clause 6.4.4
description	See clause 6.25
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1
uses	See clause 6.12

1317 **6.12 Uses statement**

1318 From RFC6020, the "uses" statement is used to reference a "grouping" definition. It takes one argument,
 1319 which is the name of the grouping.

1320 The *grouping* and *uses* statement should be handled and resolved prior to mapping the YANG to CSDL.
 1321 Since the *grouping* statement does not define a node in the schema tree, there is no YANG depiction.

1322 The YANG code shown below, uses the "endpoint" grouping defined in clause 6.11 in a definition of an
 1323 HTTP server in some other module.

```

1324 import acme-system {
1325     prefix "acme";
1326 }
1327
1328     container http-server {
1329         leaf name {
1330             type string;
1331         }
1332         uses acme:endpoint;
1333     }
    
```

1334 Table 16 shows the mapping of the *uses* statement's sub-statements.

1335

Table 16 – Uses statement mapping

Statement	Mapping
augment	See clause 6.15
description	See clause 6.25
if-feature	See clause 6.20
refine	See clause 6.12.1
reference	See clause 6.26
status	See clause 6.24
when	See clause 6.27

1336 6.12.1 Refine statement

1337 From RFC6020, some of the properties of each node in the grouping can be refined with the "refine"
 1338 statement. The argument is a string that identifies a node in the grouping. This node is called the refine's
 1339 target node.

1340 The preprocessor should which resolves to uses statement should also resolve the refine statement.

1341 In the above example, if port 80 should be the default for the HTTP server, default can be added as a
 1342 refinement.

```

1343     container http-server {
1344         leaf name {
1345             type string;
1346         }
1347         uses acme:endpoint {
1348             refine port {
1349                 default 80;
1350             }
1351         }
1352     }
  
```

1353 6.13 Rpc statement

1354 From RFC6020, the "rpc" statement is used to define a NETCONF RPC operation. It takes one
 1355 argument, which is an identifier, followed by a block of sub-statements that holds detailed rpc information.

1356 The *rpc* statement is mapped to a CSDL Action. The NETCONF RPC semantics are replaced by the
 1357 Redfish action semantics. Note, parameters can be complex

1358 6.13.1 Mapping YANG code to Redfish CSDL

1359 From the purpose of illustration, an XML instance example for rpc is shown below, from RFC6020.

```

1360 <rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
1361   <rock-the-house xmlns="http://example.net/rock">
1362     <zip-code>27606-0100</zip-code>
1363   </rock-the-house>
1364 </rpc>
  
```

1365 The YANG code for the above *rpc* example is shown below.

```

1366 rpc rock-the-house {
1367     input {
1368         leaf zip-code {
1369             type string;
1370         }
1371     }
1372 }
    
```

1373 The resultant CSDL fragment is shown below.

```

1374 <Action Name="rock-the-house" IsBound="true">
1375     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/rpc"/>
1376     <Parameter Name="rock_the_houseInput" Type="rock_the_houseInputType">
1377         <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/input"/>
1378     </Parameter>
1379 </Action>
1380
1381 <ComplexType Name="rock_the_houseInputType" >
1382     <Property Name="zip code" Type="Edm.String"/>
1383 </ComplexType>
    
```

1384 Table 17 shows the mapping of the *rpc* statement's sub-statements.

1385 **Table 17 – Rpc statement mapping**

YANG	Redfish JSON and CSDL
description	See clause 6.25
grouping	See clause 6.11
if-feature	See clause 6.20
input	See clause 6.13.2
	<Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/input"/>
output	See clause 6.13.3
	<Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/output"/>
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1

1386 **6.13.2 Input statement**

1387 See clause 6.13.1, which includes the *input* statement in the discussion.

1388 The value of Name attribute is synthesized by appending the string "Input" to the value of the *rpc*
 1389 statement. The value of the Type attribute is synthesized by appending the string "InputType" to the
 1390 value of the *rpc* statement.

1391 The "input type" shall be declared as a ComplexType.

1392 **6.13.3 Output statement**

1393 See clause 6.13.1, which includes the *output* statement in the discussion.

1394 The value of Name attribute is synthesized by appending the string "Output" to the value of the *rpc*
 1395 statement. The value of the Type attribute is synthesized by appending the string "OutputType" to the
 1396 value of the *rpc* statement.

1397 The "output type" shall be declared as a ComplexType.

```

1398 rpc rock-the-house {
1399     input {
1400         leaf zip-code {
1401             type string;
1402         }
1403     }
1404     output {
1405         leaf volume {
1406             type int16;
1407         }
1408     }
1409 }

```

1410 The value of Name attribute is synthesized by appending the string "Output" to the value of the *output*
 1411 statement.

```

1412 <Action Name="rock-the-house" IsBound="true">
1413     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/rpc"/>
1414     . . .
1415     <ReturnType Name="rock_the_houseOutput" Type="rock_the_houseOutputType">
1416         <Annotation Term="Redfish.Yang.NodeType"
1417             EnumMember ="Redfish.Yang.NodeTypes/output"/>
1418     </ReturnType>
1419 </Action>
1420
1421 <ComplexType Name="rock_the_houseOutputType" >
1422     <Property Name="volume" Type="Edm.int16"/>
1423 </ComplexType>

```

1424 6.14 Notification statement

1425 From RFC6020, the "notification" statement is used to define a NETCONF notification. It takes one
 1426 argument, which is an identifier, followed by a block of sub-statements that holds detailed notification
 1427 information.

1428 The *notification* statement is mapped to an EntityType.

1429 6.14.1 Mapping YANG code to Redfish CSDL

1430 From the purpose of illustration, an XML instance example of a notification is shown below, from
 1431 RFC6020.

```

1432 <notification
1433     xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
1434     <eventTime>2008-07-08T00:01:00Z</eventTime>
1435     <event xmlns="http://example.com/event">
1436         <event-class>fault</event-class>
1437         <reporting-entity>
1438             <card>Ethernet0</card>
1439         </reporting-entity>
1440         <severity>major</severity>
1441     </event>
1442 </notification>

```

1443 The YANG code for the above notification example is shown below.

1444
1445
1446
1447
1448
1449
1450
1451
1452

```
notification event {
  leaf event-class {
    type string;
  }
  anyxml reporting-entity;
  leaf severity {
    type string;
  }
}
```

1453 The resultant CSDL fragment is shown below.

1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477

```
<EntityType Name="event" BaseType="Resource.1.0.0.Resource">
  <Annotation Term="Redfish.Yang.NodeType"
    EnumMember ="Redfish.Yang.NodeTypes/notification"/>
  <Annotation Term="OData.Description" String=""/>
  <Annotation Term="OData.AdditionalProperties" Bool="false"/>
  <Property Name="event-class", Type="edm:String" >
    <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
    <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
    <Annotation Term="OData.Description" String=""/>
    <Annotation Term="OData.LongDescription" String=""/>
  </Property>
  <Property Name="severity", Type="edm:String" >
    <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
    <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
    <Annotation Term="OData.Description" String=""/>
    <Annotation Term="OData.LongDescription" String=""/>
  </Property>
  <Property Name="reporting-entity", Type="edm:String" >
    <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/anyxml"/>
    <Annotation Term="OData.Permissions" EnumMember="OData.Permissions/Read"/>
    <Annotation Term="OData.Description" String=""/>
    <Annotation Term="OData.LongDescription" String=""/>
  </Property>
</EntityType>
```

1478 Table 18 shows the mapping of the *notification* statement's sub-statements.

1479 **Table 18 – Notification statement mapping**

YANG	Redfish JSON and CSDL
anyxml	See clause 6.10
choice	See clause 6.9
description	See clause 6.25
grouping	See clause 6.11
if-feature	See clause 6.20
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
reference	See clause 6.26
status	See clause 6.24
typedef	See clause 6.2.1
uses	See clause 6.12

1480 **6.15 Augment statement**

1481 From RFC6020, The "augment" statement allows a module or submodule to add to the schema tree
 1482 defined in an external module, or the current module and its submodules, and to add to the nodes from a
 1483 grouping in a "uses" statement. The argument is a string that identifies a node in the schema tree.

1484 The augment statement is treated as a pre-processor directive. The resulting CSDL contains the superset
 1485 of augmentations, and also an annotation which indications what was augmented.

1486 For example, the following examples show the *augment* statement, which augments the interfaces
 1487 container.

1488 The following is an *augment* statement for an "interfaces" container.

```

1489     container interfaces {
1490         list ifEntry {
1491             key "ifIndex";
1492
1493             leaf ifIndex {
1494                 type uint32;
1495             }
1496             leaf ifDescr {
1497                 type string;
1498             }
1499             leaf ifType {
1500                 type iana:IfType;
1501             }
1502             leaf ifMtu {
1503                 type int32;
1504             }
1505         }
1506     }
  
```

1507 The following is an *augment* statement that augments the ifEntry *list* statement. In example, there is a
 1508 conditional *when* statement associated with the augment.

```

1509     import interface-module {
1510         prefix "if";
1511     }
1512     augment "/if:interfaces/if:ifEntry" {
1513         when "if:ifType='ds0'";
1514         leaf ds0ChannelNumber {
1515             type ChannelNumber;
1516         }
1517     }
  
```

1518 The resultant CSDL is shown below. Note if the augment statement adds more than one entry, then the
 1519 CSDL for each entry contains the augment annotation and conditional annotation.

```

1520 <Property Name="ifMtu" Type="Redfish.Yang.int32">
1521     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
1522     <Annotation Term="Redfish.Yang.YangType" String="int32"/>
1523     . . .
1524 </Property>
1525 <Property Name="ds0ChannelNumber" Type="Redfish.Yang.int32">
1526     <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
1527     <Annotation Term="Redfish.Yang.augment" String="if:interfaces/if:ifEntry">
1528         <Annotation Term="Redfish.Yang.when" String=" if:ifType='ds0'"/>
1529     </Annotation>
1530     <Annotation Term="Redfish.Yang.YangType" String="int32"/>
1531     . . .
1532 </Property>
  
```

1533 Table 19 shows the mapping of the *augment* statement and its sub-statements.

1534

Table 19 – Augment statement mapping

YANG	Redfish JSON and CSDL
case	See clause 6.9.3
choice	See clause 6.9
description	See clause 6.25
if-feature	See clause 6.20
leaf	See clause 6.6
leaf-list	See clause 6.7
list	See clause 6.8
reference	See clause 6.26
status	See clause 6.24
uses	See clause 6.12
when	See clause 6.27

1535 6.16 Identity statement

1536 From RFC6020, the "identity" statement is used to define a new globally unique, abstract, and untyped
 1537 identity. Its only purpose is to denote its name, semantics, and existence. An identity can either be
 1538 defined from scratch or derived from a base identity.

1539 There is no YANG depiction of an *identity* statement.

1540 The *identity* statement is mapped to a complex annotation.

1541 Identity results in ComplexType (see system RFC)

1542 6.16.1 Mapping YANG code to Redfish CSDL

1543 The general *identity* statement is shown below.

```
1544 identity [identity value]
```

1545 The resultant CSDL is shown below.

```
1546 <ComplexType Name="[identity value]">
1547   <Annotation Term="Redfish.Yang.identity" String="[identity value]" />
1548   <Annotation Term="Redfish.Yang.description" String=". . ." />
1549 </ComplexType>
```

1550 The *identity* statement from RFC7317 is shown below.

```
1551 identity authentication-method {
1552   description "Base identity for user authentication methods.";
1553 }
```

1554 The resultant CSDL is shown below:

```
1555 <ComplexType Name="authentication method">
1556   <Annotation Term="Redfish.Yang.identity" String="authentication-method" />
1557   <Annotation Term="Redfish.Yang.description"
1558     String="Base identity for user authentication methods." />
1559 </ComplexType>
```

1560 Table 20 shows the mapping of the *Identity* statement's sub-statements.

1561

Table 20 – Identity statement mapping

Statement	Mapping
base	baseType = "the base identity string". See clause 6.16.2
	<Annotation Term="Redfish.Yang.base" String="text from the yang description statement"/>
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24

1562 6.16.2 Base statement

1563 The "base" statement is optional and takes as an argument a string that is the name of an existing
 1564 identity, from which the new identity is derived. If no "base" statement is present, the identity is defined
 1565 from scratch.

1566 See clause 6.16.1 for the mapping.

1567 6.17 Extension statement

1568 From RFC6020, the "extension" statement allows the definition of new statements within the YANG
 1569 language. This new statement definition can be imported and used by other modules.

1570 The *extension* statement is mapped to an annotation. The extended statement is placed in a
 1571 Redfish.Yang.statement annotation, in which the string attribute contains the entire YANG statement

1572 `<Annotation Term="Redfish.Yang.extension" String="[text from extension statement]"/ >`

1573 The general extension statement is shown below. Note that the extended statement, along with its value,
 1574 is also shown.

```
1575 Extension [extended statement] {
1576     argument [argument value]
1577 }
1578
1579 [extended statement] [extended value];
```

1580 The resultant CSDL is shown below. The extended statement and value are used in the Annotation for
 1581 Redfish.Yang.statement.

```
1582 <EntityType ...>
1583     <Annotation Term="Redfish.Yang.extension" String="[extended statement]">
1584         <Annotation Term="Redfish.Yang.argument" String="[argument value]"/>
1585     </Annotation>
1586     <Annotation Term="Redfish.Yang.statement"
1587         String="[extended statement] [extended value]"/>
1588 </EntityType>
```

1590 Example YANG code for the *extension* statement from the MPLS OpenConfig RFC is shown below.

```
1591 extension openconfig-version {
1592     argument "semver" {
1593         yin-element false;
1594     }
1595 }
1596
1597 openconfig-version 6;
```


1598 The resultant Redfish CSDL is shown below.

```

1599 <EntityType ...>
1600   <Annotation Term="Redfish.Yang.extension" String="openconfig-version">
1601     <Annotation Term="Redfish.Yang.argument" String="semver">
1602       <Annotation Term="Redfish.Yang.yin_element"
1603         EnumMember="Redfish.Yang.YinElement/false"/>
1604     </Annotation>
1605   </Annotation>
1606   <Annotation Term="Redfish.Yang.statement" String="openconfig-version 6"/>
1607 </EntityType>
    
```

1608 Table 21 shows the mapping of the *Extension* statement's sub-statements.

1609 **Table 21 – Extension statement mapping**

Statement	Mapping
description	See clause 6.25
reference	See clause 6.26
status	See clause 6.24

1610 **6.18 Argument statement**

1611 From RFC6020, the "argument" statement, which is optional, takes as an argument a string that is the
 1612 name of the argument to the keyword.

1613 The *argument* statement is mapped to the annotation, which is within the annotation of its parent
 1614 statement.

```

1615 <Annotation Term="Redfish.Yang.argument" String="[value of argument statement]"/ >
    
```

1616 The YANG code for the argument statement from the MPLS OpenConfig RFC is shown below.

```

1617 extension [extension value] {
1618   argument [augument value] {
1619     . . .
1620   }
1621 }
    
```

1622 The resultant Redfish CSDL is shown below.

```

1623 <Annotation Terem=Redfish.Yang.extension" String="[extension value]: >
1624   <Annotation Term="Redfish.Yang.argument" String="[augument value]" >
1625     . . .
1626   </Annotation>
1627 </Annotation>
    
```

1628 Example YANG code for the argument statement from the MPLS OpenConfig RFC is shown below.

```

1629 extension openconfig-version {
1630   argument "semver" {
1631     yin-element false;
1632   }
1633 }
    
```

1634 The resultant Redfish CSDL is shown below.

```

1635 <Annotation Term="Redfish.Yang.extension" String="openconfig-version">
1636   <Annotation Term="Redfish.Yang.argument" String="semver">
1637     <Annotation Term="Redfish.Yang.yin element"
1638       EnumMember="Redfish.Yang.YinElement/false"/>
1639   </Annotation>
1640 </Annotation>
    
```

1641 Table 22 shows the mapping of the *argument* statement's sub-statements.

1642

Table 22 – Argument statement mapping

Statement	Mapping
yin-element	One of <Annotation Term="Redfish.Yang.yin_element" EnumMember="Redfish.Yang.YinElement/false"/> <Annotation Term="Redfish.Yang.yin_element" EnumMember="Redfish.Yang.YinElement/true"/>

1643 **6.19 Feature statement**

1644 From RFC6020, the "feature" statement is used to define a mechanism by which portions of the schema
 1645 are marked as conditional. A feature name is defined that can later be referenced using the "if-feature"
 1646 statement.

1647 The *feature* statement is mapped to the annotation, within the scope of its parent statement.

```
1648 <Annotation Term="Redfish.Yang.feature" String="[value of feature statement]"/ >
```

1649 The YANG code for the *feature* statement from RFC 7277 is shown below.

```
1650 feature ipv4-non-contiguous-netmasks {
1651     description "Indicates support for configuring non-contiguous subnet masks.";
1652 }
```

1653 Table 23 shows the mapping of the *Feature* statement.

1654

Table 23 – Feature statement mapping

YANG	Redfish JSON and CSDL
if-feature	See clause 6.20
status	See clause 6.24
reference	See clause 6.26

1655 **6.20 If-feature statement**

1656 From RFC6020, the "if-feature" statement makes its parent statement conditional. The argument is the
 1657 name of a feature, as defined by a "feature" statement.

1658 The *if-feature* statement is mapped to the annotation, within the annotation of its parent statement.

```
1659 <Annotation Term="Redfish.Yang.if-feature" String="[value of if-feature statement]"/ >
```

1660 The YANG code for the *if-feature* statement from RFC 7277 is shown below.

```
1661 leaf create-temporary-addresses {
1662     if-feature ipv6-privacy-autoconf;
1663     type boolean;
1664     default false;
1665     description
1666         "If enabled, the host creates temporary addresses as
1667         described in RFC 4941.";
1668     reference
1669         "RFC 4941: Privacy Extensions for Stateless Address
1670         Autoconfiguration in IPv6";
1671 }
```

1672 The resultant Redfish CSDL is shown below.

```
1673 <Property Name="create_temporary_address" Type="Redfish.Yang.boolean">
1674   <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/leaf"/>
1675   <Annotation Term="Redfish.Yang.YangType" String="boolean"/>
1676   <Annotation Term="Redfish.Yang.if-feature" String="ipv6-privacy-autoconf"/>
1677   . . .
1678 </Property>
```

1679 There are no sub-statements specified for the *if-feature* statement.

1680 6.21 Deviation statement

1681 From RFC6020, the "deviation" statement defines a hierarchy of a module that the device does not
1682 implement faithfully. The argument is a string that identifies the node in the schema tree where a
1683 deviation from the module occurs.

1684 The *deviation* statement is mapped to the annotation.

```
1685 <Annotation Term="Redfish.Yang.deviation" String="[value of deviation statement]"/ >
```

1686 The YANG code for the *deviation* statement is shown below.

```
1687 module [module value] {
1688   deviation [deviation value] {
1689     deviate [deviate value] {
1690       . . .
1691     }
1692   }
1693 }
```

1694 The resultant CSDL is show below.

```
1695 <EntityType Name="[module value]" BaseType="Resource.v1_0_0.Resource">
1696   <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/module"/>
1697   <Annotation Term="Redfish.Yang.deviation" String="[deviation value]"/ >
1698     <Annotation Term="Redfish.Yang.deviate" String="[deviate value]"/ >
1699   </Annotation>
1700   . . .
1701 </EntityType>
```

1702 The YANG code for the *deviation* statement is shown below.

```
1703 deviation /base:system/base:user/base:type {
1704   deviate add {
1705     default "admin"; // new users are 'admin' by default
1706   }
1707 }
```

1708 The resultant CSDL is show below.

```
1709 <EntityType Name="<module value>" BaseType="Resource.v1_0_0.Resource">
1710   <Annotation Term="Redfish.Yang.NodeType" EnumMember ="Redfish.Yang.NodeTypes/module"/>
1711   <Annotation Term="Redfish.Yang.deviation"
1712     String="/base:system/base:user/base:type" >
1713     <Annotation Term="Redfish.Yang.deviate" String="add" />
1714   </Annotation>
1715   . . .
1716 </EntityType>
```

1717 Table 24 shows the mapping of the *deviation* statement's sub-statements.

1718

Table 24 – Deviate statement mapping

Statement	Mapping
description	See clause 6.25
deviate	See clause 6.22
reference	See clause 6.26

1719

6.22 Deviate statement

1720 From RFC6020, The "deviate" statement defines how the device's implementation of the target node
 1721 deviates from its original definition. The argument is one of the strings "not-supported", "add", "replace",
 1722 or "delete".

1723 See clause 6.21 which shows the mapping of the *deviate* statement

1724 Table 25 shows the mapping of the *deviate* statement's sub-statements.

1725

Table 25 – Deviate statement mapping

Statement	Mapping
config	See clause 6.23
default	<Annotation Term="Redfish.Yang.default" String="text from default statement"/>
mandatory	One of <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/false"/> <Annotation Term="Redfish.Yang.mandatory" EnumMember="Redfish.Yang.Mandatory/true"/>
max-element	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=max_elements/>/true"/>
min-element	<Annotation Term="Redfish.Yang.max_elements" Redfish.Yang.uint64=min_elements/>/true"/>
must	<Annotation Term="Redfish.Yang.must" String="the XPath sting from the yang statement"> <Annotation Term="Redfish.Yang.error_message" String="text from error-message statement"/> <Annotation Term="Redfish.Yang.error_app_tag" String="text from error-app-tag statement"/> <Annotation Term="Redfish.Yang.description" String="text from description statement"/> <Annotation Term="Redfish.Yang.reference" String="text from reference statement"/> </Annotation>
type	See clause 6.4
unique	<Annotation Term="Redfish.Yang.unique" String="text from unique statement"/>
units	<Annotation Term="Redfish.Yang.units" String="text from unit statement"/>

1726

6.23 Config statement

1727 From RFC6020, the "config" statement takes as an argument the string "true" or "false". If "config" is
 1728 "true", the definition represents a configuration.

1729 The config statement is mapped to one of two annotations.

1730 <Annotation Term="Redfish.Yang.config" EnumMember="Redfish.Yang.ConfigPermission/false"/>
 1731 <Annotation Term="Redfish.Yang.config" EnumMember="Redfish.Yang.ConfigPermission/true"/>

1732 The YANG code for a *leaf* statement is shown below.

```
1733 leaf clientRequestCount {
1734     description "Client Request Count";
1735     type uint32;
1736     config "false";
1737 }
```

1738 The resultant CSDL fragment for the JSON properties is shown below.

```
1739 <Property Name="clientRequestCount" Type="Redfish.Yang.uint32">
1740     <Annotation Term="Redfish.Yang.NodeType" EnumMember="Redfish.Yang.NodeTypes/leaf"/>
1741     <Annotation Term="Redfish.Yang.YangType" String="uint32"/>
1742     <Annotation Term="Redfish.Yang.config"
1743         EnumMember="Redfish.Yang.ConfigPermission/false"/>
1744 </Property>
```

1745 The *config* statement has no sub-statements.

1746 6.24 Status statement

1747 From RFC6020, the "status" statement takes as an argument one of the strings "current", "deprecated",
1748 or "obsolete".

1749 The *status* statement is mapped to one of three annotations.

```
1750 <Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus/current" />
1751 <Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus/deprecated" />
1752 <Annotation Term="Redfish.Yang.status" EnumMember="Redfish.Yang.NodeStatus/obsolete" />
```

1753 The YANG code for a *status* statement from RFC 7224 is shown below

```
1754 identity iso88023Csmacd {
1755     base iana-interface-type;
1756     status deprecated;
1757     description "...";
1758     reference "...";
1759 }
```

1760 The resultant CSDL fragment for the JSON properties is shown below.

```
1761 <Property Name="iso88023Csmacd" Type="...">
1762     <Annotation Term="Redfish.Yang.status" String="deprecated"/>
1763     . . .
1764 </Property>
```

1765 The *status* statement has no sub-statements.

1766 6.25 Description statement

1767 From RFC6020, the "description" statement takes as an argument a string that contains a human-
1768 readable textual description of this definition.

1769 The *description* statement is mapped to the annotation.

```
1770 <Annotation Term="Redfish.Yang.description" String="[value of description statement]"/>
1771 <Annotation Term="OData.Description" String="[value from description statement]"/>
```

1772 The YANG code for a *description* statement from RFC 7224 is shown below

```
1773 identity iso88023Csmacd {
1774     base iana-interface-type;
1775     status deprecated;
1776     description "Deprecated via RFC 3635. Use ethernetCsmacd(6) instead";
1777     reference "...";
1778 }
```

1779 The resultant CSDL fragment for the JSON properties is shown below.

```
1780 <Property Name="iso88023Csmacd" Type="...">
1781   <Annotation Term="Redfish.Yang.description"
1782     String="Deprecated via RFC 3635. Use ethernetCsmacd(6) instead"/>
1783   <Annotation Term="OData.Description"
1784     String="Deprecated via RFC 3635. Use ethernetCsmacd(6) instead"/>
1785   . . .
1786 </Property>
```

1787 The *description* statement has no sub-statements.

1788 Note: The string for the LongDescription annotation is constructed from the *reference* statement. The
1789 construction adds normative text to the value of the *reference* statement, such as "The element shall ...".

1790 6.26 Reference statement

1791 The "reference" statement takes as an argument a string that is used to specify a textual cross-reference
1792 to an external document, either another module that defines related management information, or a
1793 document that provides additional information relevant to this definition.

1794 The *reference* statement is mapped to an annotation.

```
1795 <Annotation Term="Redfish.Yang.reference" String="[value of reference statement]"/ >
```

1796 The YANG code for a *reference* statement from RFC 7224 is shown below

```
1797 identity iso88023Csmacd {
1798   base iana-interface-type;
1799   status deprecated;
1800   description "...";
1801   reference "RFC 3635 - Definitions of Managed Objects for the Ethernet-like Interface
1802   Types";
1803 }
```

1804 The resultant CSDL fragment for the JSON properties is shown below.

```
1805 <Property Name="iso88023Csmacd" Type="...">
1806   <Annotation Term="Redfish.Yang.reference"
1807     String=" RFC 3635 - Definitions of Managed Objects for the Ethernet-like Interface
1808   Types"/>
1809   . . .
1810 </Property>
```

1811 The *reference* statement has no sub-statements.

1812 6.27 When statement

1813 From RFC6020, the "when" statement makes its parent data definition statement conditional. The node
1814 defined by the parent data definition statement is only valid when the condition specified by the "when"
1815 statement is satisfied. The statement's argument is an XPath expression, which is used to formally
1816 specify this condition.

1817 The *when* statement is mapped to the annotation.

```
1818 <Annotation Term="Redfish.Yang.when" String="[value of when statement]"/ >
```

1819 See clause 6.15 for an example of the YANG to CSDL mapping.

1820 The *when* statement has no sub-statements.

1821 6.28 Unmapped YANG statements

1822 If YANG code is read which does not conform the statement format, then the following annotation should
1823 be added to the resultant CSDL. This will indicate that the original YANG file should be reviewed and the
1824 source of the "statement" annotation be found.

1825 `<Annotation Term="Redfish.Yang.statement" String="text from the yang statement"/>`

1826 An example of YANG code from RFC 7317, which may cause a "statement" annotation is shown below.
1827 The nacm:default-deny-all line does not follow the statement format.

```
1828 rpc system-restart {  
1829     nacm:default-deny-all;  
1830     description  
1831         "Request that the entire system be restarted immediately.  
1832         A server SHOULD send an rpc reply to the client before  
1833         restarting the system."  
1834 }  
1835
```

1836

**ANNEX A
(informative)**

Change log

Version	Date	Description
0.1.0a	05/10/2016	Initial draft
0.2.0	05/23/2016	Incorporate the mapping from the Visio diagrams
0.3.0	05/25/2016	Clean up. Use RFC 6020 for ordering clauses. Rewrite "Lists" mapping to correspond to the DHCP collection resource construct.
0.4.0	05/29/2016	Added clauses for each YANG statement. Add cross-references in tables.
0.5.0	05/30/2016	Added clauses for each YANG sub-statement
0.5.1	06/05/2016	Revised based on June 2-3 meetings. Add examples.
0.5.2	05/06/2016	Minor fixes
0.5.3		Commented open issues.
0.5.6	10/13/2016	Modifications from the June 14 F2F review

1837
1838
1839
1840
1841

1842