



1
2
3
4

Document Identifier: DSP0218

Date: 2024-06-21

Version: 1.2.0

5
6

Platform Level Data Model (PLDM) for Redfish Device Enablement

7
8
9
10

Supersedes: 1.1.2

Document Class: Normative

Document Status: Published

Document Language: en-US

11 Copyright Notice

12 Copyright © 2019–2022, 2024 DMTF. All rights reserved.

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

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27 withdrawn or modified after publication, and shall be indemnified and held harmless by any party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

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

33 This document’s normative language is English. Translation into other languages is permitted.

CONTENTS

35	Foreword	9
36	Acknowledgments	9
37	Introduction.....	11
38	Document conventions.....	11
39	1 Scope	12
40	2 Normative references	12
41	3 Terms and definitions	14
42	4 Symbols and abbreviated terms.....	16
43	5 Conventions	17
44	5.1 Reserved and unassigned values.....	17
45	5.2 Byte ordering.....	17
46	5.3 PLDM for Redfish Device Enablement data types	17
47	5.3.1 varstring PLDM data type	18
48	5.3.2 schemaClass PLDM data type	18
49	5.3.3 nnint PLDM data type	19
50	5.3.4 bejEncoding PLDM data type	19
51	5.3.5 bejTuple PLDM data type	20
52	5.3.6 bejTupleS PLDM data type.....	20
53	5.3.7 bejTupleF PLDM data type	20
54	5.3.8 bejTupleL PLDM data type	21
55	5.3.9 bejTupleV PLDM data type.....	21
56	5.3.10 bejNull PLDM data type	22
57	5.3.11 bejInteger PLDM data type	22
58	5.3.12 bejEnum PLDM data type.....	23
59	5.3.13 bejString PLDM data type.....	23
60	5.3.14 bejReal PLDM data type.....	24
61	5.3.15 bejBoolean PLDM data type	24
62	5.3.16 bejBytestring PLDM data type	24
63	5.3.17 bejSet PLDM data type.....	25
64	5.3.18 bejArray PLDM data type.....	25
65	5.3.19 bejChoice data PLDM type	25
66	5.3.20 bejPropertyAnnotation PLDM data type	26
67	5.3.21 bejRegistryItem PLDM data type	27
68	5.3.22 bejResourceLink PLDM data type	27
69	5.3.23 bejResourceLinkExpansion PLDM data type	27
70	5.3.24 bejLocator PLDM data type	27
71	5.3.25 rdeOpID PLDM data type	28
72	6 PLDM for Redfish Device Enablement version.....	28
73	7.1 Redfish Provider architecture overview	29
74	7.1.1 Roles	29
75	7.2 Redfish Device Enablement concepts	30
76	7.2.1 RDE Device discovery and registration	30
77	7.2.2 Data instances of Redfish schemas: Resources	32
78	7.2.3 Dictionaries	36
79	7.2.4 PLDM RDE Events	54
80	7.2.5 Task support	55
81	7.3 Type code	56
82	7.4 Transport protocol type supported.....	56
83	7.5 Error completion codes	56
84	7.6 Timing specification	58
85	8 Binary Encoded JSON (BEJ)	59

86	8.1	BEJ design principles.....	59
87	8.2	SFLV tuples	60
88	8.2.1	Sequence number.....	60
89	8.2.2	Format.....	60
90	8.2.3	Length	60
91	8.2.4	Value	61
92	8.3	Deferred binding of data	61
93	8.4	BEJ encoding	63
94	8.4.1	Conversion of JSON data types to BEJ.....	64
95	8.4.2	Resource links	64
96	8.4.3	Registry items	65
97	8.4.4	Annotations	65
98	8.4.5	Choice encoding for properties that support multiple data types	66
99	8.4.6	Properties with invalid values	66
100	8.5	BEJ decoding.....	67
101	8.5.1	Conversion of BEJ data types to JSON.....	67
102	8.5.2	Annotations	68
103	8.5.3	Sequence numbers missing from dictionaries.....	69
104	8.5.4	Sequence numbers for read-only properties in modification Operations	69
105	8.5.5	Annotations for RDE Devices	69
106	8.6	Example encoding and decoding.....	70
107	8.6.1	Example dictionary.....	70
108	8.6.2	Example encoding	73
109	8.6.3	Example decoding	76
110	8.7	BEJ locators	78
111	9	Operational behaviors	79
112	9.1	Initialization (MC perspective).....	79
113	9.1.1	Sample initialization ladder diagram	79
114	9.1.2	Initialization workflow diagram	80
115	9.2	Operation/Task lifecycle.....	82
116	9.2.1	Example Operation command sequence diagrams.....	82
117	9.2.2	Operation/Task overview workflow diagrams (Operation perspective)	86
118	9.2.3	RDE Operation state machine (RDE Device perspective)	94
119	9.3	Event lifecycle	108
120	10	PLDM commands for Redfish Device Enablement.....	110
121	11	PLDM for Redfish Device Enablement – Discovery and schema commands	113
122	11.1	NegotiateRedfishParameters command (0x01) format	113
123	11.2	NegotiateMediumParameters command (0x02) format.....	115
124	11.3	GetSchemaDictionary command (0x03) format	116
125	11.4	GetSchemaURI command (0x04) format	117
126	11.5	GetResourceETag command (0x05) format.....	118
127	11.6	GetOEMCount command (0x06) format.....	119
128	11.7	GetOEMName command (0x07) format.....	120
129	11.8	GetRegistryCount command (0x08) format.....	120
130	11.9	GetRegistryDetails command (0x09) format.....	121
131	11.10	SelectRegistryVersion command (0x0A) format.....	121
132	11.11	GetMessageRegistry command (0x0B) format	122
133	11.12	GetSchemaFile command (0x0C) format	123
134	12	PLDM for Redfish Device Enablement – RDE Operation and Task commands	124
135	12.1	RDEOperationInit command (0x10) format	124
136	12.2	SupplyCustomRequestParameters command (0x11) format.....	127
137	12.3	RetrieveCustomResponseParameters command (0x12) format.....	131
138	12.4	RDEOperationComplete command (0x13) format.....	132
139	12.5	RDEOperationStatus command (0x14) format.....	133
140	12.6	RDEOperationKill command (0x15) format	136

141 12.7 RDEOperationEnumerate command (0x16) format 137

142 13 PLDM for Redfish Device Enablement – Utility commands 138

143 13.1 RDEmultipartSend command (0x30) format..... 138

144 13.2 RDEmultipartReceive command (0x31) format 140

145 14 Additional Information..... 142

146 14.1 RDE Multipart transfers..... 142

147 14.1.1 Flag usage for RDEmultipartSend 142

148 14.1.2 Flag usage for RDEmultipartReceive 143

149 14.1.3 RDE Multipart transfer examples 143

150 14.2 Implementation notes..... 145

151 14.2.1 Schema updates 145

152 14.2.2 Storage of dictionaries 145

153 14.2.3 Dictionaries for related schemas 145

154 14.2.4 [MC] HTTP/HTTPS POST Operations..... 146

155 14.2.5 Consistency checking of read Operations 146

156 14.2.6 [MC] Placement of RDE Device resources in the outward-facing Redfish URI

157 hierarchy 147

158 14.2.7 LogEntry and LogEntryCollection resources 147

159 14.2.8 On-demand pagination 148

160 14.2.9 Considerations for Redfish clients 148

161 14.2.10 OriginOfCondition in Redfish events..... 148

162 14.2.11 [MC] Merging dictionaries with OEM extensions 148

163 14.2.12 PATCH on Array Properties..... 148

164 ANNEX A (normative) Change log..... 150

165

166 **Figures**

167 Figure 1 – RDE Roles 30

168 Figure 2 – Example linking of Redfish Resource and Redfish Entity Association PDRs 34

169 Figure 3 – Schema linking without Redfish entity association PDRs 35

170 Figure 4 – Parallel Resource Linking for Metrics 36

171 Figure 5 – Dictionary binary format 40

172 Figure 6 – DummySimple schema 71

173 Figure 7 – DummySimple dictionary – binary form 73

174 Figure 8 – Example Initialization ladder diagram 80

175 Figure 9 – Typical RDE Device discovery and registration 82

176 Figure 10 – Simple read Operation ladder diagram 83

177 Figure 11 – Complex Read Operation ladder diagram 84

178 Figure 12 – Write Operation ladder diagram 85

179 Figure 13 – Write Operation with long-running Task ladder diagram 86

180 Figure 14 – RDE Operation lifecycle overview (holistic perspective) 90

181 Figure 15 – RDE Task lifecycle overview (holistic perspective) 93

182 Figure 16 – Operation lifecycle state machine (RDE Device perspective) 108

183 Figure 17 – Redfish event lifecycle overview 110

184 Figure 18 – RDEMultipartSend example 144

185 Figure 19 – RDEMultipartReceive example 145

186

187 **Tables**

188 Table 1 – PLDM for Redfish Device Enablement data types and structures 17

189 Table 2 – varstring data structure 18

190 Table 3 – schemaClass enumeration 19

191 Table 4 – nnint encoding for BEJ 19

192 Table 5 – bejEncoding data structure 19

193 Table 6 – bejTuple encoding for BEJ 20

194 Table 7 – bejTupleS encoding for BEJ 20

195 Table 8 – bejTupleF encoding for BEJ 20

196 Table 9 – BEJ format codes (high nibble: data types) 21

197 Table 10 – bejTupleL encoding for BEJ 21

198 Table 11 – bejTupleV encoding for BEJ 22

199 Table 12 – bejNull value encoding for BEJ 22

200 Table 13 – bejInteger value encoding for BEJ 23

201 Table 14 – bejEnum value encoding for BEJ 23

202 Table 15 – bejString value encoding for BEJ 23

203 Table 16 – bejString special character escape sequences 23

204 Table 17 – bejReal value encoding for BEJ 24

205 Table 18 – bejReal value encoding example 24

206 Table 19 – bejBoolean value encoding for BEJ 24

207 Table 20 – bejBytestring value encoding for BEJ 25

208 Table 21 – bejSet value encoding for BEJ 25

209 Table 22 – bejArray value encoding for BEJ 25

210 Table 23 – bejChoice value encoding for BEJ 25

211 Table 24 – bejPropertyAnnotation value encoding for BEJ 26

212 Table 25 – bejPropertyAnnotation value encoding example 26

213 Table 26 – bejRegistryItem value encoding for BEJ 27

214 Table 27 – bejResourceLink value encoding for BEJ 27

215 Table 28 – bejResourceLinkExpansion value encoding for BEJ 27

216 Table 29 – bejLocator value encoding 27

217 Table 30 – rdeOpID data structure 28

218 Table 31 – Redfish dictionary binary format 38

219 Table 32 – Dictionary entry example for a property supporting multiple formats 41

220 Table 33 – Redfish Operations 43

221 Table 34 – Redfish operation headers 45

222 Table 35 – Redfish operation request query options 50

223 Table 36 – Query parameter support requirement 51

224 Table 37 – PLDM for Redfish Device Enablement completion codes 57

225 Table 38 – HTTP codes for standard PLDM completion codes 58

226 Table 39 – Timing specification 59

227 Table 40 – Sequence number dictionary indication 60

228 Table 41 – JSON data types supported in BEJ 61

229 Table 42 – BEJ deferred binding substitution parameters 62

230 Table 43 – Message annotation related property BEJ locator encoding 66

231 Table 44 – Conditionally Required Annotations for RDE Devices 70

232 Table 45 – DummySimple dictionary (tabular form) 72

233 Table 46 – Initialization Workflow 81

234 Table 47 – Operation lifecycle overview 87

235 Table 48 – Task lifecycle overview 91

236 Table 49 – Task lifecycle state machine 95

237 Table 50 – Event lifecycle overview 108

238 Table 51 – PLDM for Redfish Device Enablement command codes 111

239 Table 52 – NegotiateRedfishParameters command format 114

240 Table 53 – NegotiateMediumParameters command format 116

241 Table 54 – GetSchemaDictionary command format 117

242 Table 55 – GetSchemaURI command format 118

243 Table 56 – GetResourceETag command format 119

244 Table 57 – GetOEMCount command format 119

245 Table 58 – GetOEMName command format 120

246 Table 59 – GetRegistryCount command format 120

247 Table 60 – GetRegistryDetails command format 121

248 Table 61 – SelectRegistryVersion command format 122

249 Table 62 – GetMessageRegistry command format 122

250 Table 63 – GetSchemaFile command format 123

251 Table 64 – RDEOperationInit command format 124

252 Table 65 – SupplyCustomRequestParameters command format 128

253 Table 66 – RetrieveCustomResponseParameters command format 132

254 Table 67 – RDEOperationComplete command format 133

255 Table 68 – RDEOperationStatus command format 133

256 Table 69 – RDEOperationKill command format 136

257 Table 70 – RDEOperationEnumerate command format 138

258 Table 71 – RDEMultipartSend command format 139
259 Table 72 – RDEMultipartReceive command format 141
260

261

Foreword

262 The *Platform Level Data Model (PLDM) for Redfish Device Enablement* (DSP0218) was prepared by the
263 PMCI (Platform Management Communications Infrastructure) Working Group of DMTF.

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

266 Acknowledgments

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

268 Editors:

- 269 • Bill Scherer – Hewlett Packard Enterprise
- 270 • Harsha Sidramappa – Hewlett Packard Enterprise
- 271 • Balaji Natrajan – Microchip Technology Inc.

272 Contributors:

- 273 • Richelle Ahlvers – Broadcom Inc.
- 274 • Jeff Autor – Hewlett Packard Enterprise
- 275 • Patrick Caporale – Lenovo
- 276 • Mike Garrett – Hewlett Packard Enterprise
- 277 • Jeff Hilland – Hewlett Packard Enterprise
- 278 • Yuval Itkin – NVIDIA Corporation
- 279 • Ira Kalman – Intel
- 280 • Deepak Kodihalli – IBM
- 281 • Eliel Louzoun – Intel
- 282 • Ben Lytle – Hewlett Packard Enterprise
- 283 • Rob Mapes – Marvell
- 284 • Balaji Natrajan – Microchip Technology Inc.
- 285 • Edward Newman – Hewlett Packard Enterprise
- 286 • Zvika Perry Peleg – Cavium
- 287 • Scott Phuong – Cisco Systems, Inc.
- 288 • Jeffrey Plank – Microchip Technology Inc.
- 289 • Joey Rainville – Hewlett Packard Enterprise
- 290 • Patrick Schoeller – Hewlett Packard Enterprise
- 291 • Hemal Shah – Broadcom Inc.
- 292 • Bob Stevens – Dell Inc.
- 293 • Richard Thomaiyar – Intel
- 294 • Bill Vetter – Lenovo

- 295 • Ryan Weldon – Marvell
- 296 • Henry Yang – Marvell

297

Introduction

298 The *Platform Level Data Model (PLDM) for Redfish Device Enablement Specification* defines messages
299 and data structures used for enabling PLDM-capable devices to participate in Redfish-based
300 management without needing to support either JavaScript Object Notation (JSON, used for operation
301 data payloads) or [Secure] Hypertext Transfer Protocol (HTTP/HTTPS, used to transport and configure
302 operations). This document specifies how to convert Redfish operations into a compact binary-encoded
303 JSON (BEJ) format transported over PLDM, including the encoding and decoding of JSON and the
304 manner in which HTTP/HTTPS headers and query options may be supported under PLDM. In this
305 specification, Redfish management functionality is divided between the three roles: the client, which
306 initiates management operations; the RDE Device, which ultimately services requests; and the
307 management controller (MC), which translates requests and serves as an intermediary between the client
308 and the RDE Device.

309 Document conventions

310 Clause naming conventions

311 While all clauses of this specification are relevant from the perspective of both MCs and RDE Devices, a
312 few clauses are primarily targeted at one or the other. This document uses the following naming
313 conventions for clauses:

- 314 • The titles of clauses that are primarily of interest to MCs are prefixed with “[MC]”.
- 315 • The titles of clauses that are primarily of interest to RDE Devices are prefixed with “[Dev]”
- 316 • Unless explicitly marked, the subclauses of a clause marked as being primarily of interest to
317 one role are also primarily of interest to that same role
- 318 • Clauses that are of primary interest to more than one role are not prefixed

319 NOTE This specification is designed such that clients have no need to be aware whether the RDE Device whose
320 data they are interacting with is supporting Redfish directly or through an MC proxy.

321 Typographical conventions

322 The following typographical conventions are used in this document:

- 323 • Document titles are marked in *italics*.

324

325
326

Platform Level Data Model (PLDM) for Redfish Device Enablement

1 Scope

This specification defines messages and data structures used for enabling PLDM devices to participate in Redfish-based management without needing to support either JavaScript Object Notation (JSON, used for operation data payloads) or [Secure] Hypertext Transfer Protocol (HTTP/HTTPS, used to transport and configure operations). This document specifies how to convert Redfish operations into a compact binary-encoded JSON (BEJ) format transported over PLDM, including the encoding and decoding of JSON and the manner in which HTTP/HTTPS headers and query options shall be supported under PLDM. This document does not specify the resources (data models) for use with RDE Devices or any details of handling the Redfish security model. Transferring firmware images is not intended to be within the scope of this specification as this function is the primary scope of [DSP0267](#), the PLDM for Firmware Update specification.

In this specification, Redfish management functionality is divided between the three roles: the client, which initiates management operations; the RDE Device, which ultimately services requests; and the management controller (MC), which translates requests and serves as an intermediary between the client and the RDE Device. Of these roles, the RDE Device and MC roles receive extensive treatment in this specification; however, the client role is no different from standard Redfish. An implementer of this specification is only required to support the features of one of the RDE Device or MC roles. In particular, an RDE Device is not required to implement MC-specific features and vice versa.

This specification is not a system-level requirements document. The mandatory requirements stated in this specification apply when a particular capability is implemented through PLDM messaging in a manner that is conformant with this specification. This specification does not specify whether a given system is required to implement that capability. For example, this specification does not specify whether a given system shall support Redfish Device Enablement over PLDM. However, if a system does support Redfish Device Enablement over PLDM or other functions described in this specification, the specification defines the requirements to access and use those functions over PLDM.

Portions of this specification rely on information and definitions from other specifications, which are identified in clause 2. Several of these references are particularly relevant:

- DMTF [DSP0266](#), *Redfish Scalable Platforms Management API Specification Redfish Scalable Platforms Management API Specification*, defines the main Redfish protocols.
- DMTF [DSP0240](#), *Platform Level Data Model (PLDM) Base Specification*, provides definitions of common terminology, conventions, and notations used across the different PLDM specifications as well as the general operation of the PLDM messaging protocol and message format.
- DMTF [DSP0245](#), *Platform Level Data Model (PLDM) IDs and Codes Specification*, defines the values that are used to represent different type codes defined for PLDM messages.
- DMTF [DSP0248](#), *Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification*, defines the event and Redfish PDR data structures referenced in this specification.

2 Normative references

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

- 368 (including any corrigenda or DMTF update versions) applies. Earlier versions may not provide sufficient
369 support for this specification.
- 370 DMTF DSP0222, *Network Controller Sideband Interface (NC-SI) Specification 1.1*,
371 https://www.dmtf.org/sites/default/files/standards/documents/DSP0222_1.1.pdf
- 372 DMTF DSP0236, *MCTP Base Specification 1.2*,
373 https://www.dmtf.org/sites/default/files/standards/documents/DSP0236_1.2.pdf
- 374 DMTF DSP0240, *Platform Level Data Model (PLDM) Base Specification 1.1*,
375 https://www.dmtf.org/sites/default/files/standards/documents/DSP0240_1.1.pdf
- 376 DMTF DSP0241, *Platform Level Data Model (PLDM) Over MCTP Binding Specification 1.0*,
377 https://www.dmtf.org/sites/default/files/standards/documents/DSP0241_1.0.pdf
- 378 DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes Specification 1.3*,
379 https://www.dmtf.org/sites/default/files/standards/documents/DSP0245_1.3.pdf
- 380 DMTF DSP0248, *Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification*
381 *1.1*, https://www.dmtf.org/sites/default/files/standards/documents/DSP0248_1.1.pdf
- 382 DMTF DSP0266, *Redfish Scalable Platforms Management API Specification 1.6*,
383 https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.6.pdf
- 384 DMTF DSP0267, *PLDM for Firmware Update Specification 1.0*,
385 https://www.dmtf.org/sites/default/files/standards/documents/DSP0267_1.0.pdf
- 386 DMTF DSP4014, *DMTF Process for Working Bodies 2.4*,
387 https://www.dmtf.org/sites/default/files/standards/documents/DSP4014_2.4.pdf
- 388 ECMA International Standard ECMA-404, *The JSON Data Interchange Syntax*, [https://www.ecma-](https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf)
389 [international.org/publications/files/ECMA-ST/ECMA-404.pdf](https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf)
- 390 IETF RFC 2781, *UTF-16, an encoding of ISO 10646*, February 2000,
391 <https://www.ietf.org/rfc/rfc2781.txt>
- 392 IETF STD63, *UTF-8, a transformation format of ISO 10646* <https://www.ietf.org/rfc/std/std63.txt>
- 393 IETF RFC 4122, *A Universally Unique Identifier (UUID) URN Namespace*, July 2005,
394 <https://www.ietf.org/rfc/rfc4122.txt>
- 395 IETF RFC 4646, *Tags for Identifying Languages*, September 2006,
396 <https://www.ietf.org/rfc/rfc4646.txt>
- 397 IETF RFC 7231, R. Fielding et al., *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*,
398 <https://tools.ietf.org/html/rfc7231>
- 399 IETF RFC 7232, R. Fielding et al., *Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests*,
400 <https://www.ietf.org/rfc/rfc7232.txt>
- 401 IETF RFC 7234, R. Fielding et al., *Hypertext Transfer Protocol (HTTP/1.1): Caching*,
402 <https://tools.ietf.org/rfc/rfc7234.txt>
- 403 ISO 8859-1, *Final Text of DIS 8859-1, 8-bit single-byte coded graphic character sets — Part 1: Latin*
404 *alphabet No.1*, February 1998
- 405 ISO/IEC Directives, Part 2, *Principles and rules for the structure and drafting of ISO and IEC documents*,
406 <https://www.iso.org/sites/directives/current/part2/index.xhtml>

407 ITU-T X.690 (08/2015), *Information technology – ASN.1 encoding rules: Specification of Basic Encoding*
408 *Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*,
409 <https://handle.itu.int/11.1002/1000/12483>

410 Open Data Protocol, <https://www.oasis-open.org/standards#odtav4.0>

411 3 Terms and definitions

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

414 The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"),
415 "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described
416 in [ISO/IEC Directives, Part 2](#), Clause 7. The terms in parentheses are alternatives for the preceding term,
417 for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that
418 [ISO/IEC Directives, Part 2](#), Clause 7 specifies additional alternatives. Occurrences of such additional
419 alternatives shall be interpreted in their normal English meaning.

420 The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as
421 described in [ISO/IEC Directives, Part 2](#), Clause 6.

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

425 Refer to [DSP0240](#) for terms and definitions that are used across the PLDM specifications, [DSP0248](#) for
426 terms and definitions used specifically for PLDM Monitoring and Control, and to [DSP0266](#) for terms and
427 definitions specific to Redfish. For the purposes of this document, the following additional terms and
428 definitions apply.

429 3.1

430 Action

431 Any standard Redfish action defined in a standard Redfish Schema or any custom OEM action defined in
432 an OEM schema extension

433 3.2

434 Annotation

435 Any of several pieces of metadata contained within BEJ or JSON data. Rather than being defined as part
436 of the major schema, annotations are defined in a separate, global annotation schema.

437 3.3

438 Client

439 Any agent that communicates with a management controller to enable a user to manage Redfish-
440 compliant systems and RDE Devices

441 3.4

442 Collection

443 A Redfish container holding an array of independent Redfish resource Members that in turn are typically
444 represented by a schema external to the one that contains the collection itself.

445 3.5

446 Device Component

447 A top-level entry point into the schema hierarchy presented by an RDE Device

- 448 **3.6**
449 **Dictionary**
450 A binary lookup table containing translation information that allows conversion between BEJ and JSON
451 formats of data for a given resource
- 452 **3.7**
453 **Discovery**
454 The process by which an MC determines that an RDE Device supports PLDM for Redfish Device
455 Enablement
- 456 **3.8**
457 **Major Schema**
458 The primary schema defining the format of a collection of data, usually a published standard Redfish
459 schema.
- 460 **3.9**
461 **Member**
462 Any of the independent resources contained within a collection
- 463 **3.10**
464 **Metadata**
465 Information that describes data of interest, such as its type format, length in bytes, or encoding method
- 466 **3.11**
467 **OData**
468 The [Open Data protocol](#), a source of annotations in Redfish, as defined by OASIS.
- 469 **3.12**
470 **OEM Extension**
471 Any manufacturer-specific addition to major schema
- 472 **3.13**
473 **Property**
474 An individual datum contained within a Resource
- 475 **3.14**
476 **RDE Device**
477 Any PLDM terminus containing an RDE Provider that requires the intervention of an MC to receive
478 Redfish communications
- 479 **3.15**
480 **RDE Provider**
481 Any RDE Device that responds to RDE Operations. See also **Redfish Provider**.
- 482 **3.16**
483 **RDE Operation**
484 The sequence of PLDM messages and operations that represent a Redfish Operation being executed by
485 an MC and/or an RDE Device on behalf of a client. See also **Redfish Operation**.

- 486 **3.17**
487 **Redfish Operation**
488 Any Redfish operation transmitted via HTTP or HTTPS from a client to an MC for execution. See also
489 **RDE Operation**.
- 490 **3.18**
491 **Redfish Provider**
492 Any entity that responds to Redfish Operations. See also **RDE Provider**.
- 493 **3.19**
494 **Registration**
495 The process of enabling a compliant RDE Device with an MC to be an RDE Provider
- 496 **3.20**
497 **Resource**
498 A hierarchical set of data organized in the format specified in a Redfish Schema.
- 499 **3.21**
500 **Schema**
501 Any regular structure for organizing one or more fields of data in a hierarchical format
- 502 **3.22**
503 **Task**
504 Any Operation for which an RDE Device cannot complete execution in the time allotted to respond to the
505 PLDM triggering command message sent from the MC and for which the MC creates standard Redfish
506 Task and TaskMonitor objects
- 507 **3.23**
508 **Triggering Command**
509 The PLDM command that supplies the last bit of data needed for an RDE Device to begin execution of an
510 RDE Operation
- 511 **3.24**
512 **Truncated**
513 When applied to a dictionary, one that is limited to containing conversion information for properties
514 supported by an RDE Device
515
- 516 **4 Symbols and abbreviated terms**
- 517 Refer to [DSP0240](#) for symbols and abbreviated terms that are used across the PLDM specifications. For
518 the purposes of this document, the following additional symbols and abbreviated terms apply.
- 519 **4.1**
520 **BEJ**
521 Binary Encoded JSON, a compressed binary format for encoding JSON data
- 522 **4.2**
523 **JSON**
524 JavaScript Object Notation

525 **4.3**
 526 **RDE**
 527 Redfish Device Enablement

528 **5 Conventions**

529 Refer to [DSP0240](#) for conventions, notations, and data types that are used across the PLDM
 530 specifications.

531 **5.1 Reserved and unassigned values**

532 Unless otherwise specified, any reserved, unspecified, or unassigned values in enumerations or other
 533 numeric ranges are reserved for future definition by DMTF.

534 Unless otherwise specified, numeric or bit fields that are designated as reserved shall be written as 0
 535 (zero) and ignored when read.

536 **5.2 Byte ordering**

537 As with all PLDM specifications, unless otherwise specified, the byte ordering of multibyte numeric fields
 538 or multibyte bit fields in this specification shall be "Little Endian": The lowest byte offset holds the least
 539 significant byte and higher offsets hold the more significant bytes.

540 **5.3 PLDM for Redfish Device Enablement data types**

541 Table 1 lists additional abbreviations and descriptions for data types that are used in message field and
 542 data structure definitions in this specification.

543 **Table 1 – PLDM for Redfish Device Enablement data types and structures**

Data Type	Interpretation
varstring	A multiformat text string per clause 5.3.1
schemaClass	An enumeration of the various schemas associated with a collection of data, encoded per clause 5.3.2
nnint	A nonnegative integer encoded for BEJ per clause 5.3.3
bejEncoding	JSON data encoded for BEJ per clause 5.3.4
bejTuple	A BEJ tuple, encoded per clause 5.3.5
bejTupleS	A BEJ Sequence Number tuple element, encoded per clause 5.3.6
bejTupleF	A BEJ Format tuple element, encoded per clause 5.3.7
bejTupleL	A BEJ Length tuple element, encoded per clause 5.3.8
bejTupleV	A BEJ Value tuple element, encoded per clause 5.3.9
bejNull	Null data encoded for BEJ per clause 5.3.10
bejInteger	Integer data encoded for BEJ per clause 5.3.11
bejEnum	Enumeration data encoded for BEJ per clause 5.3.12
bejString	String data encoded for BEJ per clause 5.3.13
bejReal	Real data encoded for BEJ per clause 5.3.14
bejBoolean	Boolean data encoded for BEJ per clause 5.3.15

Data Type	Interpretation
bejBytestring	Bytestring data encoded for BEJ per clause 5.3.16
bejSet	Set data encoded for BEJ per clause 5.3.17
bejArray	Array data encoded for BEJ per clause 5.3.18
bejChoice	Choice data encoded for BEJ per clause 5.3.19
bejPropertyAnnotation	Property Annotation encoded for BEJ per clause 5.3.20
bejRegistryItem	A Redfish Registry Message encoded for BEJ per clause 5.3.21
bejResourceLink	Resource Link data encoded for BEJ per clause 5.3.22
bejResourceLinkExpansion	Resource Link data expanded to include schema data encoded for BEJ per clause 5.3.23
bejLocator	An intra-schema locator for Operation targeting; formatted per clause 5.3.24
rdeOpID	An Operation identifier used to link together the various command messages that comprise a single RDE Operation; formatted per clause 5.3.25

544 **5.3.1 varstring PLDM data type**

545 The varstring PLDM data type encapsulates a PLDM string that can be encoded in of any of several
546 formats.

547 **Table 2 – varstring data structure**

Type	Description
enum8	stringFormat Values: { UNKNOWN = 0, ASCII = 1, UTF-8 = 2, UTF-16 = 3, UTF-16LE = 4, UTF-16BE = 5 }
uint8	stringLengthBytes Including null terminator
variable	stringData Must be null terminated

548 **5.3.2 schemaClass PLDM data type**

549 The schemaClass PLDM data type enumerates the different categories of schemas used in Redfish. RDE
550 uses 5 main classes of schemas:

- 551 • MAJOR: the main schema containing the data for a Redfish resource. This class covers the
552 vast majority of schemas for Redfish resources.
- 553 • EVENT: the standard DMTF-published event schema, for occurrences that clients may wish to
554 be notified about.
- 555 • ANNOTATION: the standard DMTF-published annotation schema that captures metadata about
556 a major schema or payload. This schemaClass shall not be used as the primary schema for
557 BEJ encodings as annotations are specially encoded alongside the primary schema.
- 558 • COLLECTION_MEMBER_TYPE: for resources that correspond to Redfish collections, this
559 class enables access to the major schema for members of that collection from the context of the
560 collection resource. (Unlike regular resources, collections in Redfish are unversioned and
561 contain multiple members.) This schemaClass shall not be used for BEJ encodings.

- 562 • ERROR: the standard DMTF-published error schema that documents an extended error when a
563 Redfish operation cannot be completed.
- 564 • REGISTRY: A device-specific collection of Redfish registry messages used for errors and
565 events. This schemaClass shall not be used as the primary schema for BEJ encodings as
566 registry items are specially encoded alongside the primary schema via the bejRegistryItem type
567 (see 5.3.21).

568 **Table 3 – schemaClass enumeration**

Type	Description
enum8	schemaType Values: { MAJOR = 0, EVENT = 1, ANNOTATION = 2, COLLECTION_MEMBER_TYPE = 3, ERROR = 4, REGISTRY = 5 }

569 **5.3.3 nnint PLDM data type**

570 The nnint PLDM data type captures the BEJ encoding of nonnegative Integers via the following encoding:

571 The first byte shall consist of metadata for the number of bytes needed to encode the numeric value in
572 the remaining bytes. Subsequent bytes shall contain the encoded value in little-endian format. As
573 examples, the value 65 shall be encoded as 0x01 0x41; the value 130 shall be encoded as 0x01 0x82;
574 and the value 1337 shall be encoded as 0x02 0x39 0x05.

575 **Table 4 – nnint encoding for BEJ**

Type	Description
uint8	Length (N) in bytes of data for the integer to be encoded
uint8	Integer data [0] (Least significant byte)
uint8	Integer data [1] (Second least significant byte)
...	...
uint8	Integer data [N-1] (Most significant byte)

576 **5.3.4 bejEncoding PLDM data type**

577 The bejEncoding PLDM data type captures an overall hierarchical BEJ-encoded block of hierarchical
578 data.

579 **Table 5 – bejEncoding data structure**

Type	Description
ver32	BEJ Version; shall be either 1.0.0 (0xF1F0F000) or 1.1.0 (0xF1F1F000) for this specification. The actual version represented shall be at least as high as the minimum version required to support any BEJ PLDM data types included in the encoding. All BEJ PLDM data types are version 1.0 unless explicitly marked as requiring a higher version. RDE devices shall not use a BEJ encoding version higher than that supported by the MC. MCs shall not use a BEJ encoding version with an RDE Device higher than the version supported by that device.
uint16	Reserved for BEJ flags
schemaClass	Defines the primary schema type for the data encoded in bejTuple below. Shall be one of MAJOR, EVENT, or ERROR.

Type	Description
bejTuple	The encoded tuple data, defined in clause 5.3.5

580 **5.3.5 bejTuple PLDM data type**

581 The bejTuple PLDM data type encapsulates all the data for a single piece of data encoded in BEJ format.

582 **Table 6 – bejTuple encoding for BEJ**

Type	Description
bejTupleS	Tuple element for the Sequence Number field, defined in clause 5.3.6 and described in clause 8.2.1
bejTupleF	Tuple element for the Format field, defined in clause 5.3.7 and described in clause 8.2.2
bejTupleL	Tuple element for the Length field, defined in clause 5.3.8 and described in clause 8.2.3
bejTupleV	Tuple element for the Value field, defined in clause 5.3.9 and described in clause 8.2.4

583 **5.3.6 bejTupleS PLDM data type**

584 The bejTupleS PLDM data type captures the Sequence Number BEJ tuple element described in clause
585 8.2.1.

586 **Table 7 – bejTupleS encoding for BEJ**

Type	Description
nnint	Sequence number indicating the specific data item contained within this tuple. The sequence number is encoded as a nonnegative integer (nnint type) and is enhanced to indicate the dictionary to which it refers. More specifically, the low-order bit of the encoded integer is metadata used to select the dictionary within which the property encoded in the tuple may be found, and shall be one of the following values: 0b: Primary schema (including any OEM extensions) dictionary as was selected in the outermost bejEncoding PLDM data type element containing this bejTupleS 1b: Annotation schema dictionary The remainder of the integer corresponds to the sequence number encoded in the dictionary. Dictionary encodings do not include the dictionary selector flag bit.

587 **5.3.7 bejTupleF PLDM data type**

588 The bejTupleF PLDM data type captures the Format BEJ tuple element described in clause 8.2.2

589 **Table 8 – bejTupleF encoding for BEJ**

Type	Description
bitfield8	Format code; the high nibble represents the data type and the low nibble represents a series of flag bits [7:4] - principal data type; see Table 9 below for values [3] - reserved flag. 1b indicates the flag is set [2] - nullable_property flag ***. 1b indicates the flag is set [1] - read_only_property_and_top_level_annotation flag **. 1b indicates the flag is set [0] - deferred_binding flag *. 1b indicates the flag is set

590 * The deferred_binding flag shall only be set in conjunction with BEJ String data and shall never be set
 591 when encoding the format of a property inside a dictionary. See clause 8.3.

592 ** The nullable property flag shall only be set when encoding the format of a property inside a dictionary.
 593 See clause 7.2.3.2.

594 *** The read_only_property_and_top_level_annotation flag has distinct meanings when in or not in the
 595 context of a dictionary. In a dictionary, it means that a property is read-only. See clause 7.2.3.2. In a BEJ
 596 encoding, it marks a nested top-level annotation. See clause 8.4.4.1. Decoding context thus uniquely
 597 determines the meaning of this flag bit.

Table 9 – BEJ format codes (high nibble: data types)

Code	BEJ Type	PLDM Type in Value Tuple Field *
0000b	BEJ Set	bejSet
0001b	BEJ Array	bejArray
0010b	BEJ Null	bejNull
0011b	BEJ Integer	bejInteger
0100b	BEJ Enum	bejEnum
0101b	BEJ String	bejString
0110b	BEJ Real	bejReal
0111b	BEJ Boolean	bejBoolean
1000b	BEJ Bytestring	bejBytestring
1001b	BEJ Choice	bejChoice
1010b	BEJ Property Annotation	bejPropertyAnnotation
1011b	BEJ Registry Item	bejRegistryItem
1100b – 1101b	Reserved	n/a
1110b	BEJ Resource Link	bejResourceLink
1111b	BEJ Resource Link Expansion	bejResourceLinkExpansion

599 **5.3.8 bejTupleL PLDM data type**

600 The bejTupleL PLDM data type captures the Length BEJ tuple element described in clause 8.2.3.

Table 10 – bejTupleL encoding for BEJ

Type	Description
nnint	Length in bytes of value tuple field

602 **5.3.9 bejTupleV PLDM data type**

603 The bejTupleV PLDM data type captures the Value BEJ tuple element described in clause 8.2.4.

604

Table 11 – bejTupleV encoding for BEJ

Type	Description
bejNull, bejInteger, bejEnum, bejString, bejReal, bejBoolean, bejBytestring, bejSet, bejArray, bejChoice, bejPropertyAnnotation, bejResourceLink, or bejResourceLinkExpansion	Value tuple element; exact type shall match that of the Format tuple element contained within the same tuple per Table 9. For example, if a tuple has 0011b (BEJ Integer) as the Format tuple element, then the data encoded in the value tuple element will be of type bejInteger.

605 **5.3.10 bejNull PLDM data type**

606 The length tuple value for bejNull data shall be zero.

607

Table 12 – bejNull value encoding for BEJ

Type	Description
(none)	No fields

608 **5.3.11 bejInteger PLDM data type**

609 Integer data shall be encoded as the shortest sequence of bytes (little endian) that represent the value in
 610 twos complement encoding. This implies that if the value is positive and the high bit (0x80) of the MSB in
 611 an unsigned representation would be set, the unsigned value will be prefixed with a new null (0x00) MSB
 612 to mark the value as explicitly positive.
 613

614

Table 13 – bejInteger value encoding for BEJ

Type	Description
uint8	Data [0] (Least significant byte of twos complement encoding of integer)
uint8	Data [1] (Second least significant byte of twos complement encoding of integer)
...	...
uint8	Data [N-1] (Most significant byte of twos complement encoding of integer)

615 **5.3.12 bejEnum PLDM data type**

616

Table 14 – bejEnum value encoding for BEJ

Type	Description
nnint	Integer value of the sequence number for the enumeration option selected

617 **5.3.13 bejString PLDM data type**

618 All BEJ strings shall be UTF-8 encoded and null-terminated.

619

Table 15 – bejString value encoding for BEJ

Type	Description
uint8	Data [0] (First character of string data)
uint8	Data [1] (Second character of string data)
...	...
uint8	Data [N-1] (Last character of string data)
uint8	Null terminator 0x00

620 The special characters that require escaping in JSON format shall also be escaped in bejString
 621 encodings, using the backslash character ('\');

622

Table 16 – bejString special character escape sequences

Character	Escape sequence
Double quote	\"
Backslash	\\
Forward slash	\/
Backspace	\b
Form feed	\f
Line feed	\n
Carriage return	\r

623 NOTE Missing escape characters will likely cause JSON text to be malformed. RDE Devices and MCs should
 624 validate correctness of BEJ String data to avoid this occurrence.

625 **5.3.14 bejReal PLDM data type**

626 BEJ encoding for *whole*, *fract*, and *exp* that represent the base 10 encoding $whole.fract \times 10^{exp}$.

627 NOTE There is no need to express special values (positive infinity, negative infinity, NaN, negative zero) because
628 these cannot be expressed in JSON.

629 **Table 17 – bejReal value encoding for BEJ**

Type	Description
nnint	Length of <i>whole</i>
bejInteger	<i>whole</i> (includes sign for the overall real number)
nnint	Leading zero count for <i>fract</i>
nnint	<i>fract</i>
nnint	Length of <i>exp</i>
bejInteger	<i>exp</i> (includes sign for the exponent)

630 In order to distinguish between the cases where the exponent is zero and the exponent is omitted
631 entirely, an omitted exponent shall be encoded with a length of zero bytes; the exponent of zero shall be
632 encoded with a single byte (of value zero). (These cases are numerically identical but visually distinct in
633 standard text-based JSON encoding.)

634 As an example, Table 18 shows the encoding of the JSON number “1.0005e+10”:

635 **Table 18 – bejReal value encoding example**

Type	Bytes	Description
nnint	0x01 0x01	Length of <i>whole</i> (1 byte)
bejInteger	0x01	<i>whole</i> (1)
nnint	0x01 0x03	leading zero count for <i>fract</i> (3)
nnint	0x01 0x05	<i>fract</i> (5)
nnint	0x01 0x01	Length of <i>exp</i> (1)
bejInteger	0x0A	<i>Exp</i> (10)

636 **5.3.15 bejBoolean PLDM data type**

637 The bejBoolean PLDM data type captures boolean data.

638 **Table 19 – bejBoolean value encoding for BEJ**

Type	Description
uint8	Boolean value { 0x00 = logical false, all other = logical true }

639 **5.3.16 bejBytestring PLDM data type**

640 The bejBytestring PLDM data type captures a generic ordered sequence of bytes. As binary data and not
641 a true string type, no null terminator should be applied.

642 **Table 20 – bejBytestring value encoding for BEJ**

Type	Description
uint8	Data [0] (First byte of string data)
uint8	Data [1] (Second byte of string data)
...	...
uint8	Data [N-1] (Last byte of string data)

643 **5.3.17 bejSet PLDM data type**

644 The bejSet PLDM data type captures a JSON Object that in turn gathers a series of properties that may
 645 be of disparate types.

646 **Table 21 – bejSet value encoding for BEJ**

Type	Description
nnint	Count of set elements
bejTuple	First set element
bejTuple	Second set element
...	...
bejTuple	N th set element (N = Count)

647 **5.3.18 bejArray PLDM data type**

648 The bejArray PLDM data type captures a JSON Array that in turn gathers an ordered sequence of
 649 properties all of a common type.

650 **Table 22 – bejArray value encoding for BEJ**

Type	Description
nnint	Count of array elements
bejTuple	First array element
bejTuple	Second array element
...	...
bejTuple	N th array element (N = Count)

651 **5.3.19 bejChoice data PLDM type**

652 The bejChoice PLDM data type captures JSON data encoded when it can be of multiple formats.
 653 Inserting the bejChoice PLDM type alerts a decoding process that multifformat data is coming up in the
 654 BEJ datastream.

655 **Table 23 – bejChoice value encoding for BEJ**

Type	Description
bejTuple	Selected option

656 **5.3.20 bejPropertyAnnotation PLDM data type**

657 The bejPropertyAnnotation PLDM data type captures the encoding of a property annotation in the form
 658 property@annotationtype.annotationname. When the bejTupleF format code is set to
 659 bejPropertyAnnotation, the sequence number bejTupleS in the outer bejTuple shall be for the annotated
 660 property. The value bejTupleV of the outer bejTuple shall be as follows:

661 **Table 24 – bejPropertyAnnotation value encoding for BEJ**

Type	Description
bejTupleS	Sequence number for annotation property name, including the schema selector bit to mark this as being from the annotation dictionary, as defined in clause 5.3.6
bejTupleF	Format for annotation data applying to the property indicated by the sequence number above, as defined in clause 5.3.7. Implementers should be aware that this format need not match the format for the annotated property.
bejTupleL	Length in bytes of data in the bejTupleV field following, as defined in clause 5.3.8
bejTupleV	Annotation data applying to the property indicated by the sequence number above, as defined in clause 5.3.9

662 As an example, Table 25 shows the encoding of the annotation:

663 “Status@Redfish.RequiredOnCreate” : false

664 **Table 25 – bejPropertyAnnotation value encoding example**

Type	Bytes	Description
bejTupleS	0x01 0x12	Sequence number for “Status” in the current schema, The low-order bit is clear to show that this sequence number is not from the annotation dictionary. Note The actual sequence number provided here is for illustrative purposes only and may not reflect the current number for “Status” in any particular dictionary
bejTupleF	0x0A	BEJ Property Annotation
bejTupleL	0x01 0x06	Length of the annotation data. The remaining entries in this table correspond to the bejTupleV entry, which in this case is the Boolean RequiredOnCreate data.
Note; The remaining rows shown in this example are collectively the bejTupleV field for the first tuple above.		
bejTupleS	0x01 0x27	Sequence number for “Redfish.RequiredOnCreate”, The low-order bit is set to mark this sequence number as being from the annotation dictionary. Note The actual sequence number provided here is for illustrative purposes only and may not reflect the current number for “Redfish.RequiredOnCreate”
bejTupleF	0x01	BEJ boolean
bejTupleL	0x01 0x01	Length of the annotation value: one byte
bejTupleV	0x00	False

665 **5.3.21 bejRegistryItem PLDM data type**

666 The bejRegistryItem PLDM data type represents a registry message item, referenced in another schema
 667 such as event, error, or message. The bejRegistryItem PLDM data type requires BEJ version 1.1.0 (see
 668 clause 5.3.4).

669 **Table 26 – bejRegistryItem value encoding for BEJ**

Type	Description
bejTupleS	The sequence number for a message item from the registry dictionary. This sequence number shall be interpreted as from the registry dictionary, NOT from the primary schema for the enclosing bejEncoding

670 **5.3.22 bejResourceLink PLDM data type**

671 The bejResourceLink PLDM data type represents the URI that links to another Redfish Resource,
 672 specified via a resource ID for the target Redfish Resource PDR. When the bejTupleF format code is set
 673 to BEJ Resource Link in BEJ-encoded data, the four bejTupleF flag bits shall each be 0b.

674 **Table 27 – bejResourceLink value encoding for BEJ**

Type	Description
nnint	ResourceID of Redfish Resource PDR for linked schema

675 **5.3.23 bejResourceLinkExpansion PLDM data type**

676 The bejResourceLinkExpansion PLDM data type captures a link to another Redfish Resource, such as a
 677 related Redfish resource, that is expanded inline in response to a \$expand Redfish request query
 678 parameter (see clause 7.2.3.11.3). When the bejTupleF format code is set to BEJ Resource Link
 679 Expansion in BEJ-encoded data, the bejTupleF flag bits must not be set.

680 **Table 28 – bejResourceLinkExpansion value encoding for BEJ**

Type	Description
nnint	ResourceID of Redfish Resource PDR for linked schema
bejEncoding	BEJ data for expanded resource

681 **5.3.24 bejLocator PLDM data type**

682 The use of BEJ locators is detailed in clause 8.7. All sequence numbers within a BEJ locator shall
 683 reference the same schema dictionary. As each of the sequence numbers is of potentially different length,
 684 reading a sequence number in a BEJ locator must be done by first reading all previous sequence
 685 numbers in the locator. As is standard for BEJ sequence number assignment, if sequence number M
 686 corresponds to an array, sequence number M + 1 (if present) will correspond to a zero-based index within
 687 the array.

688 **Table 29 – bejLocator value encoding**

Type	Description
nnint	LengthBytes Total length in bytes of the N sequence numbers comprising this locator

Type	Description
bejTupleS	Sequence number [0]
bejTupleS	Sequence number [1]
bejTupleS	Sequence number [2]
...	...
bejTupleS	Sequence number [N - 1]

689 **5.3.25 rdeOpID PLDM data type**

690 The rdeOpID PLDM data type is an Operation identifier that can be used to link together the various
 691 command messages that comprise a single RDE Operation.

692 **Table 30 – rdeOpID data structure**

Type	Description
uint16	OperationIdentifier Numeric identifier for the Operation. Operation identifiers with the most significant bit set (1b) are reserved for use by the MC when it instantiates Operations. Operation identifiers with the most significant bit clear (0b) are reserved for use by the RDE Device when it instantiates Operations in response to commands from other protocols that it chooses to make visible via RDE. The value 0x0000 is reserved to indicate no Operation.

693 **6 PLDM for Redfish Device Enablement version**

694 The version of this Platform Level Data Model (PLDM) for Redfish Device Enablement Specification shall
 695 be 1.2.0 (major version number 1, minor version number 2, update version number 0, and no alpha
 696 version).

697 In response to the GetPLDMVersion command described in [DSP0240](#), the reported version for Type 6
 698 (PLDM for Redfish Device Enablement, this specification) shall be encoded as 0xF1F2F000.

699 **7 PLDM for Redfish Device Enablement overview**

700 This specification describes the operation and format of request messages (also referred to as
 701 commands) and response messages for performing Redfish management of RDE Devices contained
 702 within a platform management subsystem. These messages are designed to be delivered using PLDM
 703 messaging.

704 Traditionally, management has been affected via myriad proprietary approaches for limited classes of
 705 devices. These disparate solutions differ in feature sets and APIs, creating implementation and
 706 integration issues for the management controller, which ends up needing custom code to support each
 707 one separately. This consumes resources both for development of the custom code and for memory in
 708 the management controller to support it. Redfish simplifies matters by enabling a single approach to
 709 management for all RDE Devices.

710 Implementing the Redfish protocol as defined by [DSP0266](#) is a big challenge when passing requests to
 711 and from devices such as network adapters that have highly limited processing capabilities and memory
 712 space. Redfish’s messages are prohibitively large because they are encoded for human readability in
 713 HTTP/HTTPS using JavaScript Object Notation (JSON). This specification details a compressed
 714 encoding of Redfish payloads that is suitable for such devices. It further identifies a common method to
 715 use PLDM to communicate these messages between a management controller and the devices that host

716 the data the operations target. The functionality of providing a complete Redfish service is distributed
717 across components that function in different roles; this is discussed in more detail in clause 7.1.1.

718 The basic format for PLDM messages is defined in [DSP0240](#). The specific format for carrying PLDM
719 messages over a particular transport or medium is given in companion documents to the base
720 specification. For example, [DSP0241](#) defines how PLDM messages are formatted and sent using MCTP
721 as the transport. Similarly, [DSP0222](#) defines how PLDM messages are formatted and sent using NC-SI
722 and RBT as the transport. The payloads for PLDM messages are application specific. The Platform Level
723 Data Model (PLDM) for Redfish Device Enablement specification defines PLDM message payloads that
724 support the following items and capabilities:

- 725 • Binary Encoded JSON (BEJ)
 - 726 – Simplified compact binary format for communicating Redfish JSON data payloads
 - 727 – Captures essential schema information into a compact binary dictionary so that it does not need
728 to be transferred as part of message payloads
 - 729 – Defined locators allow for selection of a specific object or property inside the schema's data
730 hierarchy to perform an operation
 - 731 – Encoders and decoders account for the unordered nature of BEJ and JSON properties
- 732 • RDE Device Registration for Redfish
 - 733 – A mechanism to determine the schemas the RDE Device supports, including OEM custom
734 extensions
 - 735 – A mechanism to determine parameters for limitations on the types of communication the RDE
736 Device can perform, the number of outstanding operations it can support, and other
737 management parameters
- 738 • Messaging Support for Redfish Operations via BEJ
 - 739 – Read, Update, Post, Create, Delete Operations
 - 740 – Asynchrony support for Operations that spawn long-running Tasks
 - 741 – Notification Events for completion of long-running Tasks and for other RDE Device-specific
742 happenings¹
 - 743 – Advanced operations such as pagination and ETag support

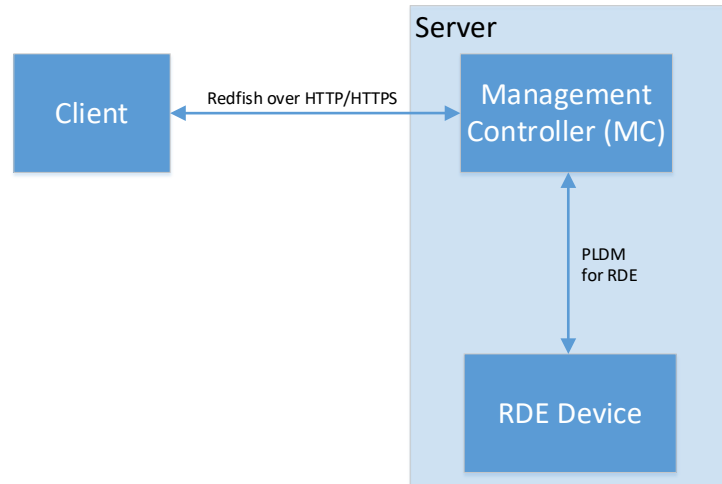
744 7.1 Redfish Provider architecture overview

745 In PLDM for Redfish Device Enablement, standard Redfish messages are generated by a Redfish client
746 through interactions with a user or a script, and communicated via JavaScript Object Notation (JSON)
747 over HTTP or HTTPS to a management controller (MC). The MC encodes the message into a binary
748 format (BEJ) and sends it over PLDM to an appropriate RDE Device for servicing. The RDE Device
749 processes the message and returns the response back over PLDM to the MC, again in binary format.
750 Next, the MC decodes the response and constructs a standard Redfish response in JSON over HTTP or
751 HTTPS for delivery back to the client.

752 7.1.1 Roles

753 RDE divides the processing of Redfish Operations into three roles as depicted in Figure 1.

1 The format for the data contained within Events is defined in [DSP0248](#). The way that events are used is defined in this specification.



754

755

Figure 1 – RDE Roles

756 The **Client** is a standard Redfish client, and needs no modifications to support operations on the data for
 757 a device using the messages defined in this specification.

758 The **MC** functions as a proxy Redfish Provider for the RDE Device. In order to perform this role, the MC
 759 discovers and registers the RDE Device by interrogating its schema support and building a representation
 760 of the RDE Device’s management topology. After this is done, the MC is responsible for receiving Redfish
 761 messages from the client, identifying the RDE Device that supplies the data relevant to the request,
 762 encoding any payloads into the binary BEJ format, and delivering them to the RDE Device via PLDM.
 763 Finally, the MC is responsible for interacting with the RDE Device as needed to get the response to the
 764 Redfish message, translating any relevant bits from BEJ back to the JSON format used by Redfish, and
 765 returning the result back to the client. The MC may also act as a client to manage RDE Devices; for this
 766 purpose, the MC may communicate directly with the RDE Device using BEJ payloads and the PLDM for
 767 Redfish Device Enablement commands detailed in this specification.

768 The **RDE Device** is an RDE Provider. To perform this role, the RDE Device must define a management
 769 topology for the resources that organize the data it provides and communicate it to the MC during the
 770 discovery and registration process. The RDE Device is also responsible for receiving Redfish messages
 771 encoded in the binary BEJ format over PLDM and sending appropriate responses back to the MC; these
 772 messages can correspond to a variety of operations including reads, writes, and schema-defined actions.

773 **7.2 Redfish Device Enablement concepts**

774 This specification relies on several key concepts, detailed in the subsequent clauses.

775 **7.2.1 RDE Device discovery and registration**

776 The processes by which an RDE Device becomes known to the MC and thus visible to clients are known
 777 as Discovery and Registration. Discovery consists of the MC becoming aware of an RDE Device and
 778 recognizing that it supports Redfish management. Registration consists of the MC interrogating specific
 779 details of the RDE Device’s Redfish capabilities and then making it visible to external clients. An example
 780 ladder diagram and a typical workflow for the discovery and registration process may be found in clause
 781 9.1.

782 **7.2.1.1 RDE Device discovery**

783 The first step of the discovery process begins when the MC detects the presence of a PLDM capable
 784 device on a particular medium. The technique by which the MC determines that a device supports PLDM

785 is outside the scope of this specification; details of this process may be found in the PLDM base
786 specification ([DSP0240](#)). Similarly, the technique by which the MC may determine that a device found on
787 one medium is the same device it has previously found on another medium is outside the scope of this
788 specification.

789 After the MC knows that a device supports PLDM, the next step is to determine whether the device
790 supports appropriate versions of required PLDM Types. For this purpose, the MC should use the base
791 PLDM GetPLDMTypes command. In order to advertise support for PLDM for Redfish Device Enablement,
792 a device shall respond to the GetPLDMTypes request with a response indicating that it supports both
793 PLDM for Platform Monitoring and Control (type 2, [DSP0248](#)) and PLDM for Redfish Device Enablement
794 (type 6, this specification). If it does, the MC will recognize the device as an RDE Device.

795 Next, the MC may use the base PLDM GetPLDMCommands command once for each of the Monitoring
796 and Control and Redfish Device Enablement PLDM Types to verify that the RDE Device supports the
797 required commands. The required commands for each PLDM Type are listed in Table 51. As with the
798 GetPLDMTypes command, use of this command is optional if the MC has some other technique to
799 understand which commands the RDE Device supports. At this point, RDE Device discovery at the PLDM
800 level is complete.

801 Once the MC has discovered the RDE Device, it invokes the NegotiateRedfishParameters command
802 (clause 11.1) to negotiate baseline details for the RDE Device. This step is mandatory unless the MC has
803 previously issued the NegotiateRedfishParameters command to the RDE Device on a different medium.
804 Baseline Redfish parameters include the following:

- 805 • The RDE Device's RDE Provider name
- 806 • The RDE Device's support for concurrency. This is the number of Operations the RDE Device
807 can support simultaneously
- 808 • RDE feature support

809 The final step in discovery is for the MC to invoke the NegotiateMediumParameters command (clause
810 11.2) in order to negotiate communication details for the RDE Device. The MC invokes this command on
811 each medium it plans to communicate with the RDE Device on as it discovers the RDE Device on that
812 medium. Medium details include the following:

813 The size of data that can be sent in a single message on the medium

814 **7.2.1.2 RDE Device registration**

815 In the registration process, the MC interrogates the RDE Device about the hierarchy of Redfish resources
816 it supports in order to act as a proxy, transparently mirroring them to external clients. The MC may skip
817 registration of the RDE Device if the PDR/Dictionary signature retrieved via the
818 NegotiateRedfishParameters command matches one previously retrieved and the MC still has the PDRs
819 and dictionaries cached.

820 In PLDM for Redfish Device Enablement, each² Redfish resource is uniquely identified by a Resource
821 Identifier that maps from the identifier to a collection of schemas that define the data for it. The identifiers
822 in turn are collected together into Redfish Resource PDRs; resources that share a common set of
823 schemas and are linked to from a common parent (such as sibling collections members) are enumerated
824 within the same PDR. Data for secondary schemas such as annotations or the message registry is linked
825 together with the major schema in the PDR structure. The resources link together to form a management
826 topology of one or more trees called device components; each resource corresponds to a node in one (or
827 more) of these trees.

² The LogEntryCollection and LogEntry resources are an exception to this; see clause 14.2.7 for a description of special handling for them.

828 The first step in performing the registration is for the MC to collect an inventory of the PDRs supported by
829 the RDE Device. There are three main PDRs of potential interest here: Redfish Resource PDRs, that
830 represent an instance of data provided by the RDE Device; Redfish Entity Association PDRs, that
831 represent the logical linking of data; and Redfish Action PDRs that represent special functions the RDE
832 Device supports. While every RDE Device must support at least one resource and thus at least one
833 Redfish Resource PDR, Redfish Action PDRs are only required if the device supports schema-defined
834 actions and Redfish Entity Association PDRs are only required under limited circumstances detailed in
835 clause 7.2.2. The MC shall collect this information by first calling the PLDM Monitoring and Control
836 GetPDRRepositoryInfo command to determine the total number of PDRs the RDE Device supports. It
837 shall then use the PLDM Monitoring and Control GetPDR command to retrieve details for each PDR from
838 the RDE Device.

839 As it retrieves the PDR information, the MC should build an internal representation of the data hierarchy
840 for the RDE Device, using parent links from the Redfish Resource PDRs and association links from the
841 Redfish Entity Association PDRs to define the management topology trees for the RDE Device.

842 After the MC has built up a representation of the RDE Device's management topology, the next step is to
843 understand the organization of data for each of the tree nodes in this topology. To this end, the MC
844 should first check the schema name and version indicated in each Redfish Resource PDR to understand
845 what the RDE Device supports. For any of these schemas, the MC may optionally retrieve a binary
846 dictionary containing information that will allow it to translate back and forth between BEJ and JSON
847 formats. It may do this by invoking the GetSchemaDictionary (clause 11.2) command with the ResourceID
848 contained in the corresponding Redfish Resource PDR.

849 NOTE While the MC may typically be expected to retrieve Redfish PDRs and dictionaries when it first registers an
850 RDE Device, there is no requirement that implementations do so. In particular, some implementations may determine
851 that one or more dictionaries supported by an RDE Device are already supported by other dictionaries the MC has
852 stored. In such a case, downloading them anew would be an unnecessary expenditure of resources.

853 After the MC has all the schema information it needs to support the RDE Device's management topology,
854 it can then offer (by proxy) the RDE Device's data up to external clients. These clients will not know that
855 the MC is interpreting on behalf of an RDE Device; from the client perspective, it will appear that the client
856 is accessing the RDE Device's data directly.

857 7.2.2 Data instances of Redfish schemas: Resources

858 In the Redfish model, data is collected together into logical groupings, called resources, via formal
859 schemas. One RDE Device might support multiple such collections, and for each schema, might have
860 multiple instances of the resource. For example, a RAID disk controller could have an instance of a disk
861 resource (containing the data corresponding to the Redfish disk schema) for each of the disks in its RAID
862 set.

863 Each resource is represented in this specification by a resource identifier contained within a Redfish
864 Resource PDR (defined in [DSP0248](#)). OEM extensions to Redfish resources are considered to be part of
865 the same resource (despite being based on a different schema) and thus do not require distinct Redfish
866 Resource PDRs.

867 Each RDE Device is responsible for identifying a management topology for the resources it supports and
868 reflecting these topology links in the Redfish Resource and Redfish Entity Association PDRs presented to
869 the MC. This topology takes the form of a directed graph rooted at one or more nodes called device
870 components. Each device component shall proffer a single Redfish Resource PDR as the logical root of
871 its own portion of the management topology within the RDE Device.

872 Links between resources can be modeled in three different ways. Direct subordinate linkage, such as
873 physical enclosure or being a component in a ComputerSystem, may be represented by setting the
874 ContainingResourceID field of the Redfish Resource PDR to the Resource ID for the parent resource. In
875 Redfish terminology, this relation is used to show subordinate resources. The parent field for the logical
876 root of a device component is set to EXTERNAL, 0x0000.

877 Logical links between resources can also be modeled. In cases where a resource and the resource to
878 which it is related are both contained within an RDE Device, these links are handled implicitly by filling in
879 the Links section of the Redfish resource when data for the resource is retrieved from the RDE Device.

880 Alternatively, logical links between resources may be represented by creating instances of Redfish Entity
881 Association PDRs (defined in [DSP0248](#)) to capture these links. In Redfish terminology, this relation is
882 used to show related resources. For example, as shown in Figure 2, the drives in a RAID subsystem are
883 subordinate to the storage controller that manages them, but are also linked to the standard Chassis
884 object. A Redfish Entity Association PDR shall only be used when a resource meets all three of the
885 following criteria:

- 886 1) The resource is contained within the RDE Device. If it is not, it does not need to be part of the
887 RDE Device's management topology model.
- 888 2) The resource is subordinate to another resource contained within the RDE Device. If it is not,
889 the resource can be linked directly to the resource outside the RDE Device by setting its parent
890 field to EXTERNAL.
- 891 3) The resource needs to be linked to another resource outside the RDE Device.

892 7.2.2.1 Alignment of resources

893 While determining how to lay out the Redfish Resource PDRs for an RDE Device may seem to be a
894 daunting task at first glance, it is actually relatively straightforward. By examining the Links section of the
895 various schemas that the RDE Device needs to support, one will see that the tree hierarchy for them is
896 already defined. Simply put, then, the RDE Device manufacturer will set up one PDR per resource or
897 group of sibling resources that share the same schema definitions and reflect the same parentage trees
898 for the PDRs as is already present for the resources in their corresponding Redfish schema definitions.

899 NOTE For collections, the RDE Device shall offer one PDR for the collection as a whole and one PDR for each set of
900 sibling entries within the collection. This is necessary to enable the MC to use the correct dictionary when encoding
901 data for a Create operation applied to an empty collection.

902 7.2.2.2 Example linking of PDRs within RDE Devices

903 This clause presents examples of the way an RDE Device can link Redfish Resource PDRs together to
904 present its data for management.

905 The example in Figure 2 models a simple rack-mounted server with local RAID storage. In this example,
906 we see a Redfish Resource PDR offering an instance of the standard Redfish Storage resource, with
907 ResourceID 123. This PDR has ContainingResourceID (abbreviated ContainingRID in the figure) set to
908 EXTERNAL as the RDE Device should be subordinate to the Storage Collection under ComputerSystem.

909 NOTE It is up to the MC to make final determinations as to where resources should be added within the Redfish
910 hierarchy. While general guidance may be found in clause 14.2.6, the technique by which MCs may ultimately make
911 such decisions is out of scope for this specification.

912 The StorageController has two Redfish Resource PDRs that list it as their container: one that offers data
913 in the VolumeCollection resource and one that offer data for four Disk resources. Finally, the PDR that
914 offers VolumeCollection resource is marked as the container for a Redfish Resource PDR that offers data
915 for the Volume resource.

916 The connections discussed so far are all direct parent linkages in the Redfish Resource PDRs because
917 the links they represent are the direct subordinate resource links from the standard Redfish storage
918 model. However, the Redfish storage model also includes notations that drives are related to (contained
919 within) a volume and that drives are related to (present inside) a chassis. These resource relations can be
920 modeled using Redfish Entity Association PDRs if the MC is managing the links. Alternatively, they can
921 be implicitly managed by the RDE Device. In this case, the RDE Device will expose the links itself by
922 filling in a Links section of the relevant resource data with references to the linked resources. While the

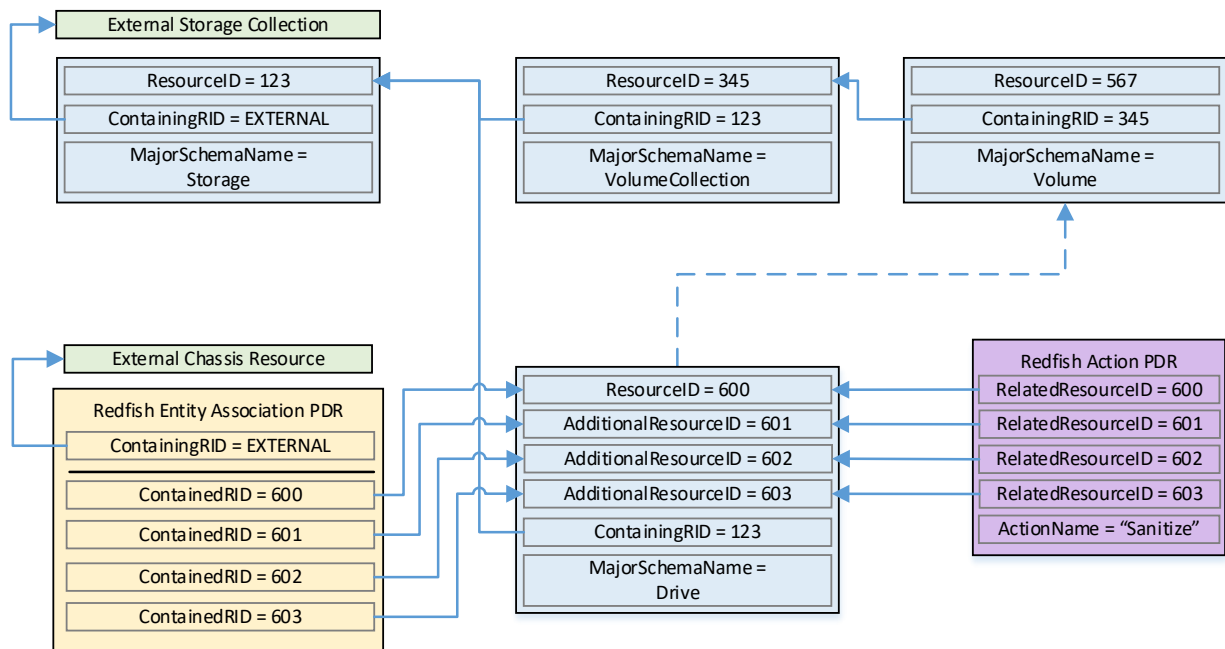
923 RDE Device could in theory provide a Redfish Entity Association PDR for this case, it serves no purpose
 924 for the MC.

925 In general, a Redfish Entity association PDR should be used when a resource is subordinate to another
 926 resource within the RDE Device but must also be linked to from another resource external to the RDE
 927 Device.

928 In the example in Figure 2, the relation between the drives and the outside Chassis resource is
 929 promulgated with a Redfish Entity Association PDR. This PDR lists the four drives as the four
 930 ContainingResourceIDs for the association, marking them to be contained within the chassis. The
 931 ContainingResourceID for this relation contains the value EXTERNAL, to show that the drives are visible
 932 outside the resource hierarchy maintained by the RDE Device. By contrast, the linkage between the
 933 drives and the Volume resource is implicitly maintained by the RDE Device. This is shown in the figure via
 934 the dashed arrows.

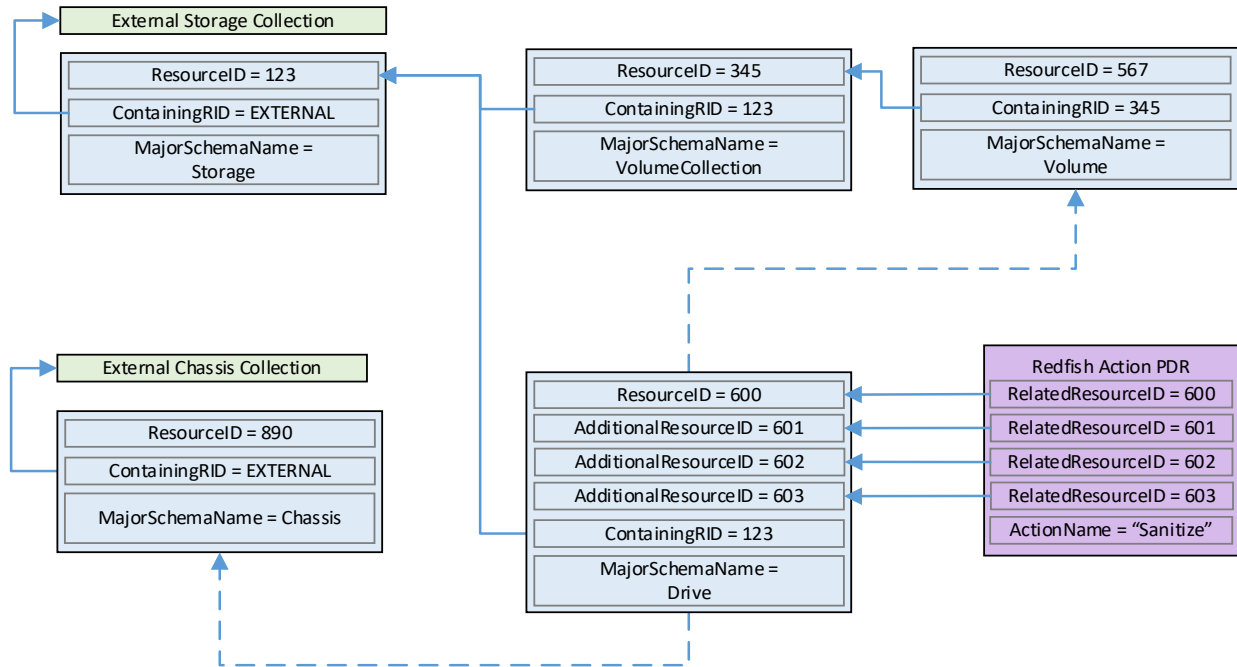
935 Finally, each of the drives supports a Sanitize operation. This is shown by instantiating a Redfish Action
 936 PDR naming the Sanitize action and linking it to each of the drives.

937 As an alternative to the PDR layout of Figure 2, in Figure 3 the RDE Device exposes its own chassis
 938 resource (labeled as Resource ID 890) rather than having the drives be part of an external chassis. The
 939 PDR for this chassis resource shows ContainingResourceID EXTERNAL to demonstrate that it belongs in
 940 the system chassis collection resource. With this modification, the links between the chassis resource and
 941 the drives can be managed internally by the RDE Device and hence no Redfish Entity Association PDR is
 942 necessary.



943

944 **Figure 2 – Example linking of Redfish Resource and Redfish Entity Association PDRs**



945

946

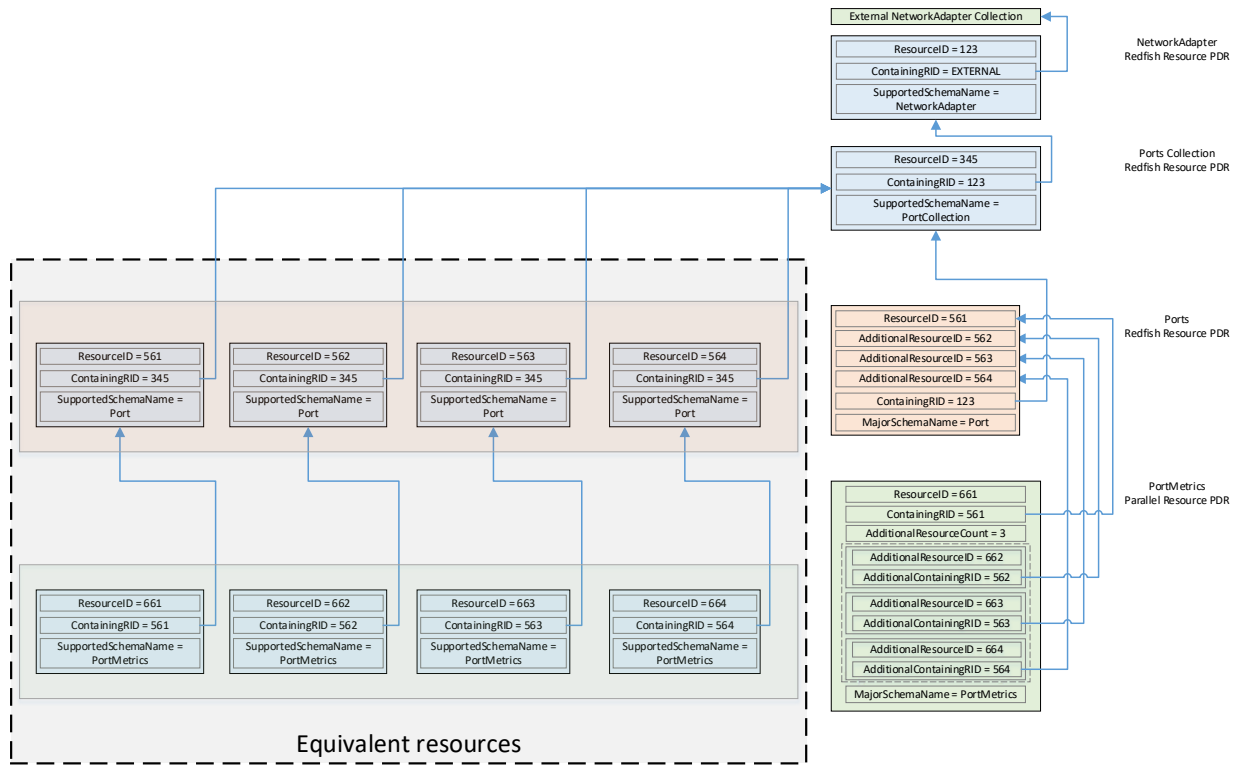
Figure 3 – Schema linking without Redfish entity association PDRs

947 **7.2.2.3 Parallel resource relationships**

948 A special case occurs when each member of a collection contains a subordinate resource of a common
 949 resource type (such as Metrics resources subordinate to Port resources). In this case, the Redfish
 950 Parallel Resource PDR (defined in [DSP0248](#)) enables each of the subordinate resources to be linked to
 951 its corresponding parent resource. This PDR shall only be used if all of the subordinate resources share
 952 in common the same major schema and the same list of OEM extension schemas. To use this PDR, each
 953 of the subordinate resources should be linked to its corresponding parent resource via the
 954 AdditionalResourceID and AdditionalContainingResourceID fields in the PDR.

955 For example, consider a series of PortMetric resources corresponding to a collection of four Ports for a
 956 NetworkAdapter resource. As shown in Figure 4 below, a typical implementation would consist of a total
 957 of at least ten Redfish resources: one NetworkAdapter, one PortCollection, four Port and four PortMetrics.
 958 As explained in the previous clause, the four Port resources can be collapsed into a single Redfish
 959 Resource PDR using the AdditionalResourceID fields to duplicate resource information from the first port
 960 to the remaining three. This is shown in orange in the figure.

961 For the PortMetrics resources, they cannot be collapsed into a single Redfish Resource PDR because
 962 each instance of the PortMetrics resource has a different parent to which it is subordinate: its
 963 corresponding Port resource. This is what the Redfish Parallel Resource PDR enables. As shown in
 964 green in the figure, we assign the ResourceID field in the Redfish Parallel Resource PDR to the
 965 ResourceID for the PortMetrics resource subordinate to port 1. The AdditionalResourceIDCount would be
 966 three (since the port 1 is already covered). Each of the three additional ports is now detailed in the
 967 Redfish Parallel Resource PDR, with an AdditionalResourceID for the PortMetrics resource and an
 968 AdditionalContainingResourceID for the Port resource to which it is subordinate. All remaining fields in the
 969 Redfish Parallel Resource PDR are the same as in the Redfish Resource PDR and are filled out in the
 970 same manner, completing this example.



971

972

Figure 4 – Parallel Resource Linking for Metrics

973 **7.2.3 Dictionaries**

974 In standard Redfish, data is encoded in JSON. In this specification, data is encoded in Binary Encoded
 975 JSON (BEJ) as defined in clause 8. In order to translate between the two encodings, the MC uses a
 976 schema lookup table that captures key metadata for fields contained within the schema. The dictionary is
 977 necessary because some of the JSON tokens are omitted from the BEJ encoding in order to achieve a
 978 level of compactness necessary for efficient processing by RDE Devices with limited memory and
 979 computational resources. In particular, the names of properties and the string values of enumerations are
 980 skipped in the BEJ encoding.

981 Each Redfish resource PDR can reference up to four classes of dictionaries for the schemas it can use³:

- 982 • Standard Redfish data schema (aka the major schema)
- 983 • Standard Redfish Event schema
- 984 • Standard Redfish Annotation schema
- 985 • Standard Redfish Error schema

986 Major and Event Dictionaries may be augmented to contain OEM extension data as defined in the
 987 Redfish base specification, [DSP0266](#).

988 Event, Error, and Annotation Dictionaries shall be common to all resources that an RDE Device provides.

³ The COLLECTION_MEMBER_TYPE schema class from clause 5.3.2 is not represented in the PDR. It can be retrieved on demand by the MC from the RDE Device via the GetSchemaDictionary command of clause 11.3.

989 Dictionaries for standard Redfish schemas are published on the DMTF Redfish website at
 990 <http://redfish.dmtf.org/dictionaries>. Naturally, these dictionaries do not include OEM extensions. RDE
 991 Devices may support their resources either with

992 the standard dictionaries or with custom dictionaries that may include OEM extensions, and that may also
 993 be truncated to contain only entries for properties supported by the RDE Device.

994 7.2.3.1 Canonizing a schema into a dictionary

995 In Redfish schemas, the order of properties is indeterminate and properties are identified by name
 996 identifiers that are of unbounded length. While this is beneficial from a human readability perspective,
 997 from a strict information-theoretical point of view, using long strings for this purpose is grossly inefficient: a
 998 numeric value of $\text{Log}_2(n\text{Children})$ bits ought to be sufficient. To make this work in practice, we impose a
 999 canonical ordering that assigns each property or enumeration value a numeric sequence number.
 1000 Sequence numbers shall be assigned according to the following rules:

- 1001 1) The children properties (properties immediately contained within other properties such as sets
 1002 or arrays) shall collectively receive an independent set of sequence numbers ranging from zero
 1003 to $N - 1$, where N is the number of children. Sequence numbers for properties that do not share
 1004 a common parent are not related in any way.
- 1005 2) For the initial revision of a Redfish schema (usually v1.0), sequence numbers shall be assigned
 1006 according to a strict alphabetical ordering of the property names from the schema.
- 1007 3) In order to preserve backward compatibility with earlier versions of schemas, for subsequent
 1008 revisions of Redfish schemas, the sequence numbers for child properties added in that revision
 1009 shall be assigned sequence numbers N to $N + A - 1$, where N is the number of sequence
 1010 numbers assigned in the previous revision and A is the number of properties added in the
 1011 present revision. (In other words, we append to the existing set and use sequence numbers
 1012 beginning with the next one available.) The new sequence numbers shall be assigned
 1013 according to a strict alphabetical ordering of their names from the schema.
- 1014 4) In the event that a property is deleted from a schema, its sequence number shall not be reused;
 1015 the sequence number for the deleted property shall forever remain allocated to that property.
- 1016 5) As with properties, the values of an enumeration shall collectively receive an independent set of
 1017 sequence numbers ranging from zero to $N - 1$, where N is the number of enumeration values.
 1018 Sequence numbers for enumeration values not belonging to the same enumeration are not
 1019 related in any way.
- 1020 6) For the initial version of a Redfish schema, sequence numbers for enumeration values shall be
 1021 assigned according to a strict alphabetical ordering of the enumeration values from the schema.
- 1022 7) In order to preserve backward compatibility with earlier versions of schemas, for subsequent
 1023 revisions of Redfish schemas, the sequence numbers for enumeration values added in that
 1024 revision shall be assigned sequence numbers N to $N + A - 1$, where N is the number of
 1025 sequence numbers assigned in the previous revision and A is the number of enumeration
 1026 values added in the present revision. The new sequence numbers shall be assigned according
 1027 to a strict alphabetical ordering of their value strings from the schema.
- 1028 8) In the event that an enumeration value is deleted from a schema, its sequence number shall not
 1029 be reused; the sequence number for the deleted enumeration value shall forever remain
 1030 allocated to that enumeration value.

1031 After the sequence numbers for properties and enumeration values are assigned, they shall be
 1032 collected together with other information from the Redfish and OEMs schema to build a dictionary in
 1033 the format detailed in clause 7.2.3.2. For every Redfish Resource PDR the RDE Device offers, it shall
 1034 maintain a dictionary that it can send to the MC on demand in response to a GetSchemaDictionary
 1035 command (clause 11.2).

1036 NOTE Rules 2 and 3 above imply that schema child properties may not be in strict alphabetical order. For example,
 1037 suppose a property node in a schema started with child fields “red”, “orange”, and “yellow” in version 1.0. Because
 1038 this is the initial version, the fields would be alphabetized: “orange” would get sequence number 0; “red”, 1; and
 1039 “yellow” would get 2. If version 1.1 of the schema were to add “blue” and “green”, they would be assigned sequence
 1040 numbers 3 and 4 respectively (because that is the alphabetical ordering of the new properties). The initial three
 1041 properties retain their original sequence numbers.

1042 For all custom dictionaries, including all truncated dictionaries, the sequence numbers listed for
 1043 standard Redfish schema properties supported by the RDE Device shall match the sequence
 1044 numbers for those same properties from the standard dictionary. This allows MCs to potentially
 1045 merge related dictionaries from RDE Devices that share a common class.

1046 Sequence numbers for array elements shall be assigned to match the zero-based index of the array
 1047 element.

1048 NOTE The ordering rules provided in this clause apply to dictionaries only. In particular, data encoded in either JSON
 1049 or BEJ format is by definition unordered.

1050 **7.2.3.2 Dictionary binary format**

1051 The binary format of dictionaries shall be as follows. All integer fields are stored little endian:

1052 **Table 31 – Redfish dictionary binary format**

Type	Dictionary Data
uint8	VersionTag Dictionary format version tag: 0x00 for DSP0218 v1.0.0, v1.1.0, v1.1.1
bitfield8	DictionaryFlags Flags for this dictionary: [7:1] - reserved for future use [0] - truncation_flag; if 1b, the dictionary is truncated and provides entries for a subset of the full Redfish schema
uint16	EntryCount Number N of entries contained in this dictionary
uint32	SchemaVersion Version of the Redfish schema encapsulated in this dictionary, in standard PLDM format. 0xFFFFFFFF for an unversioned schema. The version of the schema may be read from the filename of the schema file.
uint32	DictionarySize Size in bytes of the dictionary binary file. This value can be used as a safeguard to compare the various offsets given in subsequent fields against: buffer overruns can be avoided by validating that the offsets remain within the binary dictionary space.
bejTupleF	Format [0] Entry 0 property format. The read_only_property_and_top_level_annotation flag in the bejTupleF structure shall be set if the property is annotated as read only in the Redfish schema. The nullable_property in the bejTupleF structure shall be set if the property is annotated as nullable in the Redfish schema.
uint16	SequenceNumber [0] Entry 0 property sequence number
uint16	ChildPointerOffset [0] Entry 0 property child pointer offset in bytes from the beginning of the dictionary. Shall be 0x0000 if Format [0] is not one of {BEJ Set, BEJ Array, BEJ Enum and BEJ Choice} or in cases where a set or array contains no children elements.

Type	Dictionary Data
uint16	ChildCount [0] Entry 0 child count; shall be 0x0000 if Format [0] is not one of {BEJ Set, BEJ Array, BEJ Enum}. For a BEJ Array, the child count shall be expressed as 1.
uint8	NameLength [0] Entry 0 property/enumeration value name string length. Name length, including null terminator, shall be a maximum of 255 characters. Shall be 0x00 for an anonymous format option of a BEJ Choice-formatted property or for anonymous array entries.
uint16	NameOffset [0] Entry 0 property name string offset in bytes from the beginning of the dictionary. Shall be 0x0000 for an anonymous format option of a BEJ Choice-formatted property or for anonymous array entries.
...	...
bejTupleF	Format [N – 1] Entry (N – 1) property format. The <code>read_only_property_and_top_level_annotation</code> flag in the <code>bejTupleF</code> structure shall be set if the property is annotated as read only in the Redfish schema. The <code>nullable_property</code> in the <code>bejTupleF</code> structure shall be set if the property is annotated as nullable in the Redfish schema.
uint16	SequenceNumber [N – 1] Entry (N – 1) property sequence number
uint16	ChildPointerOffset [N – 1] Entry (N – 1) property child pointer offset in bytes from the beginning of the dictionary. Shall be 0x0000 if Format [N – 1] is not one of {BEJ Set, BEJ Array, BEJ Enum and BEJ Choice}.
uint16	ChildCount [N – 1] Entry (N – 1) child count; shall be 0x0000 if Format [N] is not one of {BEJ Set, BEJ Array, BEJ Enum}. For a BEJ Array, the child count shall be expressed as 1.
uint8	NameLength [N – 1] Entry (N – 1) property/enumeration value name string length. Name length, including null terminator, shall be a maximum of 255 characters. Shall be 0x00 for an anonymous format option of a BEJ Choice-formatted property or for anonymous array entries.
uint16	NameOffset [N – 1] Entry (N – 1) property name string offset in bytes from the beginning of the dictionary. Shall be 0x0000 for an anonymous format option of a BEJ Choice-formatted property or for anonymous array entries.
strUTF-8	Name [0] Entry 0 property name string (not present for children nodes of BEJ Choice format properties or anonymous array entries)
...	...
strUTF-8	Name [N – 1] Entry (N – 1) property name string (not present for children nodes of BEJ Choice format properties or anonymous array entries)
uint8	CopyrightLength Dictionary copyright statement string length. Copyright, including null terminator, shall be a maximum of 255 characters. May be 0x00 in which case the Copyright field below shall be omitted.
strUTF-8	Copyright Copyright statement for the dictionary. Shall be omitted if CopyrightLength is 0.

1053 Intuitively, the dictionary binary format may be thought of as a header (orange) followed by an array of
 1054 entry data (blue) followed by a table of the strings (green) naming the properties and enumeration values
 1055 for the entries. Figure 5 displays this data in graphical format:

1056

DWORD	Byte offset			
	+0	+1	+2	+3
00	VersionTag 0x00	DictionaryFlags	EntryCount ₂	EntryCount ₁
01	SchemaVersion ₄	SchemaVersion ₃	SchemaVersion ₂	SchemaVersion ₁
02	DictionarySize ₄	DictionarySize ₃	DictionarySize ₂	DictionarySize ₁
03	Format[0]	SequenceNumber[0] ₂	SequenceNumber[0] ₁	ChildPointerOffset[0] ₂
04	ChildPointerOffset[0] ₁	ChildCount[0] ₂	ChildCount[0] ₁	NameLength[0]
05	NameOffset[0] ₂	NameOffset[0] ₁
06
...	Format[N-1]	SequenceNumber[N-1] ₂	SequenceNumber[N-1] ₁	ChildPointerOffset[N-1] ₂
...	ChildPointerOffset[N-1] ₁	ChildCount[N-1] ₂	ChildCount[N-1] ₁	NameLength[N-1]
...	NameOffset[N-1] ₂	NameOffset[N-1] ₁	Name[0] ₁ *	Name[0] ₂ *
...	Name[0] ₃ *	...	Name[0] _{terminator} *	...
...
...	Name[N-1] ₁ *	Name[N-1] ₂ *	Name[N-1] ₃ *	...
...	Name[N-1] _{terminator} *	CopyrightLength	Copyright ₁	...
...	Copyright _{terminator}			

1057 **Figure 5 – Dictionary binary format**

1058 * Name strings will not be present in the dictionary for anonymous format options of BEJ Choice-
 1059 formatted properties or for anonymous array entries.

1060 **7.2.3.2.1 Hierarchical organization of entries**

1061 Within this binary format, the entries shall be sorted into clusters representing a breadth-first traversal of
 1062 the hierarchy presented by a schema. Each cluster shall in turn consist of all the sibling nodes contained
 1063 within a common parent, sorted by sequence number per the rules defined in clause 7.2.2.3 above. An
 1064 example of this organization may be found in clause 8.6.1.

1065 NOTE While not mandatory, it is acceptable that multiple dictionary entries may point to a common complex subtype
 1066 to allow reuse of that information and reduce the overall size of the dictionary. For example, Resource.status is
 1067 commonly used multiple times within the same schema, so having a single offset for it can trim some length from the
 1068 dictionary.

1069 **7.2.3.3 Properties that support multiple formats**

1070 For properties that support multiple formats, the dictionary shall contain an entry linking the property
 1071 name string to the BEJ Choice format. This choice entry shall in turn link to a series of anonymous child
 1072 entries (name offset = 0x0000) that are of the various data formats supported by the property. For
 1073 example, if a TCP/IP hostname property supports both string (“www.dmtf.org”) and numeric (the 32-bit
 1074 equivalent of 72.47.235.184) values, the dictionary might contain rows such as the following:

1075 **Table 32 – Dictionary entry example for a property supporting multiple formats**

Row	Sequence Number	Format	Name	Child Pointer
...
15	0	choice	“hostname”	18
...
18	0	string	null	null
19	1	integer	null	null
...

1076 NOTE Following the rules for sequence number assignment (see clause 7.2.3.1), each cluster of properties
 1077 contained within a given set and each cluster of enumeration values are numbered separately. Hence sequence
 1078 numbers may be repeated within a dictionary.

1079 An exception to this rule is that properties that support null and exactly one other data format shall be
 1080 collapsed into a single entry in the dictionary listing only the non-null data format. The nullable_property
 1081 bit in the bejTupleF value of the format entry in the dictionary shall be set to 1b in this case. This case is
 1082 common in the standard Redfish schemas, where most properties are nullable. This is flagged with the
 1083 “nullable” keyword in the CSDL schemas, but in the JSON schemas, it manifests as the supported type
 1084 list for the property consisting of NULL and either a solitary second type or a collection of strings that form
 1085 an enumeration.

1086 **7.2.3.4 Annotation dictionary format**

1087 Standard Redfish annotations are derived from three sources: the Redfish, odata, and message
 1088 schemas. The annotations that can be part of a JSON payload are collected together into the redfish-
 1089 payload-annotations.vX.Y.Z.json schema file. This clause details special notes that apply to building the
 1090 annotation dictionary:

- 1091 • The dictionary entries for properties in the annotation dictionary shall include the entire name of
 1092 the annotation, beginning with the ‘@’ sign and including both the annotation source (one of
 1093 redfish, message, or odata) and the annotation’s name itself. For example, the dictionary Name
 1094 field for the @odata.id property shall be an offset to the string “@odata.id”.
- 1095 • The dictionary entries for patternProperties in the annotation dictionary shall be stripped of the
 1096 wildcard patterns before the ‘@’ sign and of the trailing ‘\$’ sign but shall otherwise be treated
 1097 identically to standard properties. For example, the dictionary Name field for the “^[a-zA-Z][a-
 1098 zA-Z0-9_]*”?@Message.ExtendedInfo\$” patternProperty shall be an offset to the string
 1099 “@Message.ExtendedInfo”.
- 1100 • In accordance with the rules presented in clause 7.2.2.3, the top-level entries for annotations
 1101 (those containing the names of the annotations themselves) shall be sorted alphabetically
 1102 together for the initial version of the schema’s dictionary, and shall be appended to the list with
 1103 each schema revision. Stated explicitly, the annotations from the properties and

1104 patternProperties shall be comingled together within the entries for each revision of the
1105 dictionary.

- 1106 • Dictionary entries for children properties of annotations, such as the anonymous string value
1107 array entries for @Redfish.AllowableValues shall be structured and formatted per the rules
1108 presented in clause 7.2.2.3.

1109 7.2.3.5 Registry dictionary format

1110 Redfish messages are used in multiple places, including annotations, events, and errors. The actual
1111 message data may be retrieved from any of the various message registries including standard Redfish
1112 and OEM registries. These messages are referred to by name as the value of a string field in hosting
1113 schemas, so names such as “NetworkDevice.1.0.LinkFlapDetected” appear in BEJ-encoded JSON data
1114 for previous versions of this specification. To reduce the size of such encodings, RDE version 1.1
1115 introduces the notion of a Registry dictionary that can be referenced via the bejRegistryItem encoding
1116 format. Replacing the message name with a sequence number in the Registry dictionary achieves a
1117 reduction in encoded data for messages. This clause details special notes that apply to building the
1118 registry dictionary:

- 1119 • The registry dictionary shall consist of a top-layer set named “registry”
 - 1120 – Entries within the set shall be named for each of the registry items supported by the RDE
1121 Device. The full odata name for these entries shall be incorporated in the dictionary, and
1122 they shall be sorted lexicographically.
 - 1123 – The type of the registry items shall be bejString, and they shall be flagged as read-only.
- 1124 • Both full and truncated registry dictionaries are permitted.
- 1125 • MCs shall not attempt to merge registry dictionaries from different devices or dictionaries
1126 retrieved from the same device at different times.
- 1127 • If using the DMTF dictionary builder tool (see clause 7.2.3.7), see the tool documentation for
1128 information on how to build the registry dictionary for a device.
- 1129 • Schema entries that correspond to registry items shall be encoded in dictionaries as being of
1130 type bejString, not bejRegistryItem. This ensures backward compatibility with earlier versions of
1131 the RDE specification

1132 7.2.3.6 Links between schemas

1133 Links in Redfish schemas, identifiable as entries with Odata type odata.id, shall be represented in
1134 dictionaries as entries with format = bejString. As described in clause 8.4.2, runtime encoding of Odata
1135 links may be performed via any of bejString (with deferred bindings), bejResourceLink, or (for expansion)
1136 bejResourceLinkExpansion. This is a special case wherein a valid encoding may differ from the type
1137 specified in the dictionary.

1138 7.2.3.7 Actions in dictionaries

1139 Actions in Redfish schemas are detailed in a manner prescribed by OData that differs from how regular
1140 resource properties are presented. In contrast to the items under Redfish objects, which are written as
1141 named Property elements in CSDL, the items under actions are written as either a named Parmeter or an
1142 anonymous Return Type element.

1143 When encoding actions in RDE dictionaries, the action itself shall be encoded as an object (BEJ Set).
1144 Action parameters shall be encoded in the manner described Section 7.2.3.2, as if they were properties
1145 within the set represented by the action. The flags in the bejTupleF tuple member for action parameters
1146 shall have the following interpretation:

- 1147 • Bit 0, the deferred binding flag, shall be set to 0b for all parameters.

- 1148 • Bit 1, the read only flag, shall be set to 0b for all parameters.
- 1149 • Bit 2, the nullable property flag, shall be set to 0b for a mandatory parameter and 1b for an
- 1150 optional parameter. In Redfish CSDL schema, a parameter is mandatory if it is flagged with the
- 1151 following annotation:
- 1152 • All other bits shall be set to 0b.

1153 If the action does not support any parameters, the set for the action shall be empty (have zero children)

1154 within the dictionary encoding.

1155 The action's Return Type, if present, shall be encoded in the dictionary as an element of the type

1156 matching the specified Redfish Return Type and named "Return Type". The flags in the bejTupleF tuple

1157 member for the Return Type element shall be set as follows:

- 1158 • Bit 0, the deferred binding flag, shall be set to 0b.
- 1159 • Bit 1, the read only flag, shall be set to 1b.
- 1160 • Bit 2, the nullable property flag, shall be set to 0b.
- 1161 • All other bits shall be set to 0b.

1162 If an action does not contain a Return Type, the named Return Type element shall not be encoded into the

1163 dictionary for that action.

1164 **7.2.3.8 Building dictionaries**

1165 Available online at <https://github.com/DMTF/RDE-Dictionary>, the RDE dictionary builder automates the

1166 process of building an RDE dictionary from CSDL formatted schemas.

1167 It supports standard Redfish schemas, standalone OEM schemas, and OEM extensions to standard

1168 Redfish schemas and can build full or truncated dictionaries. For more information about installation,

1169 usage and examples of using the dictionary builder, refer to the README.md file at the above

1170 URL.Redfish Operation support.

1171 Redfish Operations are sent from a client to a Redfish Provider that is able to process them and respond

1172 appropriately. These operations are encoded in JSON and transported via either the HTTP or the HTTPS

1173 protocol.

1174 In this specification, the MC is the Redfish Provider to which the client sends operations. However, rather

1175 than responding directly, the MC is a proxy that conveys these operations to the RDE Devices that

1176 maintain the data and can provide responses to client requests. The proxied operations (that are

1177 transmitted to the RDE Device as RDE Operations) are encoded in BEJ (clause 8) and transported via

1178 PLDM. The MC, in its role as proxy Redfish Provider for the RDE Devices, translates the JSON/HTTP(S)

1179 requests from the client into BEJ/PLDM for the RDE Device, and then translates the BEJ/PLDM response

1180 from the RDE Device into a JSON/HTTP(S) response for the client.

1181 **7.2.3.9 Primary Operations**

1182 There are seven primary Redfish Operations. These are summarized in Table 33.

1183 **Table 33 – Redfish Operations**

Operation	Verb	Description
Read	GET	Retrieve data values for all properties contained within a resource.
Update	PATCH	Write updates to properties within a resource. May be to the entire resource, to a subtree rooted at any point within the resource, or to a leaf node.
Replace	PUT	Write replacements for all properties within a resource.

Create	POST	Append a new set of child data to a collection (array).
Delete	DELETE	Remove a set of child data from a collection.
Action	POST	Invoke a schema-defined Redfish action.
Head	HEAD	Retrieve just headers for the data contained in a schema.

1184 The only Redfish Operation that is required to be supported in RDE is Read; however, it is expected that
 1185 implementations will support Update as well. Create and Delete are conditionally required for RDE
 1186 Devices that contain collections; Action is conditionally required for RDE Devices that support Redfish
 1187 schema-defined actions. The Head and Replace Redfish Operations are strictly optional.

1188 **7.2.3.9.1 HTTP/HTTPS and Redfish**

1189 A full discussion of the HTTP/HTTPS protocol is beyond the scope of this specification; however, a
 1190 minimalist overview of key concepts relevant to Redfish Device Enablement follows. Readers are directed
 1191 to [DSP0266](#) for more detailed information on the usage of HTTP and HTTPS with Redfish and to
 1192 standard documentation for more general information on the HTTP/HTTPS protocols themselves.

1193 **7.2.3.9.1.1 Redfish Operation requests**

1194 Every Redfish request has a target URI to which it should be applied; this URI is the target of the
 1195 HTTP/HTTPS verb listed in Table 33. The URI may consist of several parts of interest for purposes of this
 1196 specification: a prefix that points to the RDE Device being managed, a subpath within the RDE Device
 1197 management topology, a specific resource selection preceded by an octothorp character (#), and one or
 1198 more query options preceded by a question mark (?) character.

1199 Many, but not all, Redfish requests have a JSON payload associated with them. For example, a POST
 1200 operation to create a new child element in a collection would normally contain a JSON payload for the
 1201 data being supplied for that new child element.

1202 Finally, every Redfish HTTP/HTTPS request will contain a series of headers, each of which modifies it in
 1203 some fashion.

1204 **7.2.3.9.1.2 Redfish Operation responses**

1205 The response to a Redfish HTTP/HTTPS request will also contain several elements. First, the response
 1206 will contain a status code that represents the result of the operation. Like for requests, [DSP0266](#) defines
 1207 several response headers that may need to be supplied in conjunction with a Redfish response. Finally, a
 1208 JSON payload may be present such as in the case of a read operation.

1209 **7.2.3.9.1.3 Generic handling of Redfish Operations**

1210 Generically, to handle processing of a Redfish HTTP/HTTPS request, the MC will typically implement the
 1211 following steps. This overview ignores error conditions, timeouts, and long-lived Tasks. A much more
 1212 detailed treatment may be found in clause 9.

- 1213 1) Parse the prefix of the supplied URI to pinpoint the RDE Device that the operation targets.
- 1214 2) Parse the RDE Device portion of the URI to identify the specific place in the RDE Device's
1215 management topology targeted by the operation.
- 1216 3) Identify the Redfish Resource PDR that represents that portion of the data.
- 1217 4) Using the HTTP/HTTPS verb and other request information, determine the type of Redfish
1218 operation that the client is trying to perform.
- 1219 5) Translate any request headers (clause 7.2.3.10) and query options (clause 7.2.3.11) into
1220 parameters to the corresponding PLDM request message(s).

- 1221 6) Translate the JSON payload, if present, into a corresponding BEJ (clause 8) payload for the
1222 request, using a dictionary appropriate for the target Redfish Resource PDR.
- 1223 7) Send the PLDM for Redfish Device Enablement RDEOperationInit command (clause 12.1) to
1224 begin the Operation.
- 1225 8) Send any BEJ payload to the RDE Device via one or more PLDM for Redfish Device
1226 Enablement RDEMultipartSend commands (clause 13.1) unless it was small enough to be
1227 inlined in the RDEOperationInit command.
- 1228 9) Send any request parameters to the RDE Device via the PLDM for Redfish Device Enablement
1229 SupplyCustomRequestParameters command (clause 12.2).
- 1230 10) If there was a payload but no request parameters, send the RDEOperationStatus command
1231 (clause 12.5).
- 1232 11) Retrieve and decode any BEJ-encoded JSON data for any Operation response payloads via
1233 one or more PLDM for Redfish Device Enablement RDEMultipartReceive commands (clause
1234 13.2).
- 1235 12) Retrieve any response parameters via the PLDM for Redfish Device Enablement
1236 RetrieveCustomResponseHeaders command (clause 12.3).
- 1237 13) Send the PLDM for Redfish Device Enablement RDEOperationComplete command (clause
1238 12.4) to inform the RDE Device that it may discard any data structures associated with the
1239 Task.
- 1240 14) Translate the BEJ response payload, if present, into JSON format for return to the client, using
1241 an appropriate dictionary.
- 1242 15) Prepare and send the final response to the client, adding the various HTTP/HTTPS response
1243 headers (clause 7.2.3.10) appropriate to the type of Redfish operation that was just performed.

1244 **7.2.3.10 Redfish operation headers**

1245 Several HTTP/HTTPS transport layer headers modify Redfish operations when translated in the context
1246 of RDE Operations. These are summarized in Table 34. Implementation notes for how the MC and RDE
1247 Device shall support some of these modifiers – when attached to Redfish operations – may be found in
1248 the indicated subsections. For headers not listed here, the implementation is outside the scope of this
1249 specification; implementers shall refer to [DSP0266](#) and standard HTTP/HTTPS documentation for more
1250 information on processing these headers.

1251 **Table 34 – Redfish operation headers**

Header	Clause	Where Used	Description
Request Headers			
If-Match	7.2.3.10.1	Request	If-Match shall be supported on PUT and PATCH requests for resources for which the RDE Device returns ETags, to ensure clients are updating the resource from a known state.
If-None-Match	7.2.3.10.2	Request	If this HTTP header is present, the RDE Device will only return the requested resource if the current ETag of that resource does not match the ETag sent in this header. If the ETag specified in this header matches the resource’s current ETag, the status code returned from the GET will be 304.
Custom HTTP/HTTPS Headers	7.2.3.10.3	Request and Response	Non-standard headers used for custom purposes.
Response Headers			

Header	Clause	Where Used	Description
Etag	7.2.3.10.4	Response	An identifier for a specific version of a resource, often a message digest.
Link	7.2.3.10.5	Response	Link headers shall be returned as described in the clause on Link Headers in DSP0266 .
Location	7.2.3.10.6	Response	Indicates a URI that can be used to request a representation of the resource. Shall be returned if a new resource was created.
Cache-Control	7.2.3.10.7	Response	This header shall be supported and is meant to indicate whether a response can be cached or not
Allow	7.2.3.10.8	Response	Shall be returned with a 405 (Method Not Allowed) response to indicate the valid methods for the specified Request URI. Should be returned with any GET or HEAD operation to indicate the other allowable operations for this resource.
Retry-After	7.2.3.10.9	Response	Used to inform a client how long to wait before requesting the Task information again.

1252 **7.2.3.10.1 If-Match request header**

1253 The MC shall support the If-Match header when applied to Redfish HTTP/HTTPS PUT and PATCH
1254 operations; support for other Redfish operations is optional.

1255 The parameter for this header is an ETag.

1256 In order to support this header, the MC shall convey the supplied ETag to the RDE Device via the
1257 ETag[0] field of the PLDM SupplyCustomRequestParameters command (clause 12.2) request message
1258 and supply the value ETAG_IF_MATCH for the ETagOperation field of the same message. For this
1259 header, the MC shall supply the value 1 for the ETagCount field of the request message.

1260 When the RDE Device receives an ETAG_IF_MATCH within the ETagOperation field in the
1261 SupplyCustomRequestParameters command, it shall verify that the ETag matches the current state of the
1262 targeted schema data instance before proceeding with the RDE Operation. In the event of a mismatch, it
1263 shall respond to the SupplyCustomRequestParameters command with completion code
1264 ERROR_ETAG_MATCH.

1265 In the event that both an If-Match and If-None-Match request header are supplied by the client, the MC
1266 shall respond with HTTP status code 400 – Bad Request – to the client and stop processing the request.
1267 The MC shall not send such a malformed request to the RDE Device.

1268 **7.2.3.10.2 If-None-Match request header**

1269 The MC may optionally support the If-None-Match header when applied to Redfish HTTP/HTTPS GET
1270 and HEAD operations.

1271 The parameter for this header is a comma-separated list of ETags.

1272 In order to support this header, the MC shall convey the supplied ETag(s) to the RDE Device via the
1273 ETag[i] fields of the PLDM SupplyCustomRequestParameters command (clause 12.2) request message
1274 and supply the value ETAG_IF_NONE_MATCH for the ETagOperation field of the same message. For
1275 this header, the MC shall supply the value N for the ETagCount field of the request message where N is
1276 the number of entries in the comma-separated list.

1277 When the RDE Device receives an ETAG_IF_NONE_MATCH within the ETagOperation field in the
1278 SupplyCustomRequestParameters command, it shall verify that none of the supplied ETags matches the
1279 current state of the targeted schema data instance before proceeding with the RDE Operation. In the

1280 event of a match, it shall respond to the SupplyCustomRequestParameters command with completion
1281 code ERROR_ETAG_MATCH.

1282 In the event that both an If-Match and If-None-Match request header are supplied by the client, the MC
1283 shall respond with HTTP status code 400 – Bad Request – to the client and stop processing the request.
1284 The MC shall not send such a malformed request to the RDE Device.

1285 7.2.3.10.3 Custom HTTP headers

1286 The MC shall support custom headers when applied to any Redfish HTTP/HTTPS operation. For
1287 purposes of this specification, an RDE custom header shall be considered as one with a prefix “PLDM-
1288 RDE-“. Unless explicitly specified in this specification, no standard handling is described for RDE custom
1289 headers either in this specification or in [DSP0266](#). All discussion of custom headers in this specification
1290 shall be restricted to HTTP/HTTPS custom headers of this form.

1291 The parameters for custom headers will vary by actual header type.

1292 In order to support RDE custom headers, the MC shall bundle them (including the PLDM-RDE prefix) into
1293 the request message for an invocation of the SupplyCustomRequestParameters command (clause 12.2).
1294 To do so, the MC shall set the HeaderComponent request parameter to the number of custom request
1295 parameters. For each RDE custom request parameter n , the MC shall set HeaderName[n] and
1296 HeaderParameter[n] to the name and value of the request parameter, respectively. Custom headers other
1297 than those prefixed “PLDM-RDE-“ shall not be supplied to RDE Devices in this manner.

1298 When the RDE Device receives RDE custom request parameters, it may perform any custom handling for
1299 the parameter. If it does not support a specific RDE custom request parameter received, the RDE Device
1300 shall respond with the ERROR_UNRECOGNIZED_CUSTOM_HEADER completion code.

1301 Similarly, when the RDE Device has custom response parameters to send back to a client, it shall set the
1302 HaveCustomResponseParameters flag in the OperationExecutionFlags response field of the
1303 RDEOperationInit, SupplyCustomRequestParameters, or RDEOperationStatus command to ask the MC
1304 to retrieve these parameters. Then, in response to the RetrieveCustomResponseParameters command
1305 (clause 12.3), the RDE Device shall set the ResponseHeaderCount field to the number of custom
1306 response headers it wants to send back to the client. For each custom response parameter n , the RDE
1307 Device shall set HeaderName[n] and HeaderParameter[n] to the name and value of the response
1308 parameter, respectively.

1309 Following completion of the main Operation, the MC shall check the HaveCustomResponseParameters
1310 flag in the OperationExecutionFlags response field to see if the RDE Device is supplying custom
1311 response headers (which should have a PLDM-RDE prefix). If the flag is set (with value 1b), the MC shall
1312 use the RetrieveCustomResponseParameters command (clause 12.3) to recover them from the RDE
1313 Device. The MC shall then append the recovered headers to the Redfish Operation response.

1314 7.2.3.10.3.1 PLDM-RDE-Expand-Type

1315 The MC may optionally support use of the PLDM-RDE-Expand-Type header when it receives a Redfish
1316 HTTP/HTTPS GET operation with the \$expand query option (see clause 7.2.3.11.3) to convey the
1317 expansion type parameter to the RDE Device.

1318 The parameter for this header is the type of expansion to be used in the expansion, one of
1319 EXPAND_DOT (“.”), EXPAND_TILDE (“~”), or EXPAND_STAR (“*”) and shall match the parameter given
1320 as the value of the \$expand query option. If this header is not supplied to the RDE Device, expansion
1321 shall default to type EXPAND_DOT. If no expansion type is supplied to the \$expand query option, the MC
1322 may either send this header with the default type (EXPAND_DOT) or omit it.

1323 7.2.3.10.4 ETag response header

1324 The MC shall provide an ETag header in response to every Redfish HTTP/HTTPS GET or HEAD
1325 operation.

1326 The parameter for this header is an ETag.

1327 In order to support this header, the RDE Device shall generate a digest of the schema data instance after
1328 each modification to the data in accordance with [RFC 7232](#). When the MC begins a GET or HEAD
1329 operation to the RDE Device via a PLDM RDEOperationInit command (clause 12.1), the RDE Device
1330 shall populate the ETag field in the response message to the command where the RDE Operation has
1331 completed (one of RDEOperationInit, SupplyCustomRequestParameters, or RDEOperationStatus) with
1332 this digest.

1333 When it receives an ETag field in the response message for a completed RDE Operation, the MC shall
1334 then populate this header with the digest it receives.

1335 7.2.3.10.5 Link response header

1336 The MC shall provide one or more Link headers in response to every Redfish HTTP/HTTPS GET and
1337 HEAD operation as described in [DSP0266](#).

1338 The parameter for this header is a URI.

1339 This header has three forms as described in [DSP0266](#); all three shall be supported by MCs. The handling
1340 for these three forms is detailed in the next three clauses.

1341 No special action is needed on the part of an RDE Device to support any form of the link response
1342 header.

1343 7.2.3.10.5.1 Schema form

1344 The MC shall provide a link header with “rel=describedby” to provide a schema link for the data that is or
1345 would be returned in response to a Redfish HTTP/HTTPS GET or HEAD operation. The MC may obtain
1346 this link in any of several manners:

- 1347 • An @odata.context annotation in read data may contain the schema reference.
- 1348 • The MC may have the schema reference cached.
- 1349 • The MC may retrieve the schema reference directly from the PDR encapsulating the instance of
1350 the schema data by invoking the PLDM GetSchemaURI command (clause 11.4).

1351 An example of a schema form link header is as follows; readers are referred to [DSP0266](#) for more detail:

1352

Link: </redfish/v1/JsonSchemas/ManagerAccount.v1_0_2.json>; rel=describedby

1353 7.2.3.10.5.2 Annotation form

1354 The MC should provide a link header to provide an annotation link for the data that is or would be
1355 returned in response to a Redfish HTTP/HTTPS GET or HEAD operation. The MC may obtain this link in
1356 any of several manners:

- 1357 • The MC may inspect annotations to determine whether @odata or @Redfish annotations are
1358 used.
- 1359 • The MC may retrieve the schema reference directly from the PDR encapsulating the instance of
1360 the schema data by invoking the PLDM GetSchemaURI command (clause 11.4)

1361 An example of an annotation form link header is as follows; readers are referred to [DSP0266](#) for more
1362 detail:

1363 Link: <<http://redfish.dmtf.org/schemas/Settings.json>>

1364 7.2.3.10.5.3 Passthrough form

1365 The MC shall translate link annotations returned from the RDE Device in response to a Redfish
1366 HTTP/HTTPS GET operation into link headers. In this form, the MC shall also include the schema path to
1367 the link.

1368 An example of a passthrough form link header is as follows; readers are referred to [DSP0266](#) for more
1369 detail:

1370 Link: </redfish/v1/AccountService/Roles/Administrator>; path=/Links/Role

1371 7.2.3.10.6 Location response header

1372 The MC shall provide a Location header in response to every Redfish HTTP/HTTPS POST that effects a
1373 successful create operation. The MC shall also provide a Location header in response to every Redfish
1374 Operation that spawns a long-running Task when executed as an RDE Operation.

1375 The parameter for this header is a URI.

1376 In order to support this header for completed create operations, the RDE Device shall populate the
1377 NewResourceID response parameter in the response message for the
1378 RetrieveCustomResponseParameters command (clause 12.3) with the Resource ID of the newly created
1379 collection element. Upon receipt, the MC shall combine this resource ID with the topology information
1380 contained in the Redfish Resource PDRs for the targeted PDR up through the device component root to
1381 create a local URI portion that it shall then combine with its external management URI for the RDE Device
1382 to build a complete URI for the newly added collection element. The MC shall then populate this header
1383 with the resulting URI.

1384 In order to support this header for Redfish Operations that spawn long-running Tasks when executed as
1385 RDE Operations, the MC shall generate a TaskMonitor URL for the Operation and populate the Location
1386 header with the generated URL. See clause 7.2.5 for more details.

1387 7.2.3.10.7 Cache-Control response header

1388 The MC shall provide a Cache-Control header in response to every Redfish HTTP/HTTPS GET or HEAD
1389 operation.

1390 In order to support this header for HTTP/HTTPS GET operations, the RDE Device shall mark the
1391 CacheAllowed flag in the OperationExecutionFlags field of the response message for the triggering
1392 command for the read or head Operation with an indication of the caching status of data read.

1393 When the MC reads the CacheAllowed flag in the OperationExecutionFlags field of the response
1394 message for a completed RDE Operation, it shall populate the Cache-Control response header with an
1395 appropriate value. Specifically, if the RDE Device indicates that the data is cacheable, the MC shall
1396 interpret this as equivalent to the value "public" as defined in [RFC 7234](#); otherwise, the MC shall interpret
1397 this as equivalent to the value "no-store" as defined in [RFC 7234](#).

1398 7.2.3.10.8 Allow response header

1399 The MC shall provide an Allow header in response to every Redfish HTTP/HTTPS operation that is
1400 rejected by the RDE Device specifically for the reason of being a disallowed operation, giving the
1401 ERROR_NOT_ALLOWED completion code (clause 7.5). The MC shall additionally provide an Allow
1402 response header in response to every GET (or HEAD, if supported) Redfish operation.

1403 In order to support this header, when the RDE Device responds to an RDE command with
 1404 ERROR_NOT_ALLOWED, or in response to a GET or HEAD Redfish operation, it shall populate the
 1405 PermissionFlags field of its response message with an indication of the operations that are permitted.

1406 When the MC reads the PermissionFlags field of the response message for a completed RDE Operation,
 1407 the MC shall populate this header with the supplied information.

1408 **7.2.3.10.9 Retry-After response header**

1409 The MC shall provide a Retry-After header in response to every non-HEAD Redfish Operation that when
 1410 conveyed to the RDE Device results in any transient failure (ERROR_NOT_READY; see clause 7.5).

1411 The parameter for this header is the length of time in seconds the client should wait before retrying the
 1412 request.

1413 When the RDE Device needs to defer an RDE Operation, it shall return ERROR_NOT_READY in
 1414 response to the RDEOperationInit command that begins the Operation. The RDE Device must now
 1415 choose whether to supply a specific deferral timeframe or to use the default deferral timeframe. To specify
 1416 a specific deferral timeframe, the RDE Device shall also set the HaveCustomResponseParameters flag in
 1417 the OperationExecutionFlags response field of the RDEOperationInit command to inform the MC that it
 1418 should retrieve deferral information. Then, if it did set the HaveCustomResponseParameters flag, in
 1419 response to the RetrieveCustomResponseParameters command (clause 12.3), the RDE Device shall set
 1420 the DeferralTimeframe and DeferralUnits parameters appropriately to indicate how long it is requesting
 1421 the client to wait before resubmitting the request.

1422 As an alternative to specifying a deferral timeframe via the response message for
 1423 RetrieveCustomResponseParameters, the RDE Device may skip setting the
 1424 HaveCustomResponseParameters flag in the OperationExecutionFlags response field of the
 1425 RDEOperationInit command to request that the MC supply a default deferral timeframe on its behalf.

1426 When it receives the response to the RDEOperationInit command, the MC shall check the
 1427 HaveCustomResponseParameters flag in the OperationExecutionFlags response field to see if the RDE
 1428 Device has an extended response. If the flag is set (with value 1b), the MC shall use the
 1429 RetrieveCustomResponseParameters command (clause 12.3) to recover the deferral timeframe from the
 1430 DeferralTimeframe and DeferralUnits fields of the response message. If the flag was not set, or if the RDE
 1431 Device supplied an unknown deferral timeframe (0xFF), the MC shall use a default value of 5 seconds. It
 1432 shall then populate this header with the deferral value.

1433 Both the MC and RDE Device shall be prepared for possibility that the client may retry the operation
 1434 before this deferral timeframe elapses: Operations can be re-initiated by impatient end users.

1435 **7.2.3.11 Redfish Operation request query options**

1436 In addition to HTTP/HTTPS headers, the standard Redfish management protocol defines several query
 1437 options that a client may specify in a URI to narrow the request in Redfish GET Operations. For any query
 1438 option not listed here, the MC may support it in a fashion as described in [DSP0266](#).

1439 **Table 35 – Redfish operation request query options**

Query Option	Clause	Description	Example
\$skip	7.2.3.11.1	Integer indicating the number of Members in the Resource Collection to skip before retrieving the first resource.	_____
\$top	7.2.3.11.2	Integer indicating the number of Members to include in the response.	_____

\$expand	7.2.3.11.3	Expand schema links, gluing data together into a single response. Collection: Collection by name * = all links . = all but those in Links	
\$levels	7.2.3.11.4	Qualifier on \$expand; number of links to expand out	_____
\$select	7.2.3.11.5	Top-level or a qualifier on \$expand; says to return just the specified properties	_____
excerpt	7.2.3.11.6	Returns a subset of the resource's properties that match the defined Excerpt schema annotation.	
\$filter	n/a	Limit results of a READ operation to a subset of the resource collection's members based on a \$filter expression that follows the OData-Protocol Specification.	n/a: \$filter is not supported in this specification
only	n/a	Applies to resource collections. If the target resource collection contains exactly one member, clients can use this query parameter to return that member's resource.	n/a: only is not supported in this specification

1440 Support requirements for query parameters are described in Table 36.

1441 **Table 36 – Query parameter support requirement**

Query Option	RDE Device	MC
\$skip	Optional	Should support
\$top	Optional	Should support
\$expand	Optional	Should support
\$levels	Optional	May support
\$select	Optional	May support
\$filter	Should not support	May support

1442 **7.2.3.11.1 \$skip query option**

1443 The MC should support \$skip query options when provided as part of a target URI for a Redfish
1444 HTTP/HTTPS GET operation.

1445 The parameter for this query option is an integer representing the number of members of a resource
1446 collection to skip over. See [DSP0266](#) for more details on the usage of \$skip.

1447 To support this query option, the MC shall supply the \$skip parameter in the CollectionSkip field of the
1448 SupplyCustomRequestParameters (clause 12.2) request message. In the event that this query option is
1449 not supplied as part of the target URI for an HTTP/HTTPS GET operation, the MC shall supply a value of
1450 zero in this field if it otherwise needs to supply extended request parameters; it shall not send the
1451 SupplyCustomRequestParameters just to supply a value of zero for the CollectionSkip field.

1452 When processing an RDE read Operation for a resource collection, the RDE Device shall check the
1453 CollectionSkip parameter from the SupplyCustomRequestParameters request message to determine the
1454 number of members to skip over in its response, per [DSP0266](#). In the event that the MC did not indicate
1455 the presence of extended request parameters, the RDE Device shall interpret this as a CollectionSkip
1456 value of zero. If the parameter for \$skip equals the number of elements in the collection, the RDE Device
1457 shall return an empty list. If the parameter for \$skip exceeds the number of elements in the collection, the
1458 RDE Device shall return ERROR_OPERATION_FAILED and, in accordance with the Redfish standard
1459 [DSP0266](#) respond with an annotation specifying that the value is invalid (see
1460 QueryParameterOutOfRange in the Redfish base message registry).

1461 **7.2.3.11.2 \$stop query option**

1462 The MC should support \$stop query options when provided as part of the target URI for a Redfish
1463 HTTP/HTTPS GET operation.

1464 The parameter for this query option is an integer representing the number of members of a resource
1465 collection to return. See [DSP0266](#) for more details on the usage of \$stop. If the parameter for \$stop
1466 exceeds the remaining number of members in a resource collection, the number returned shall be
1467 truncated to those remaining. For a \$stop value of zero, the response shall consist of an empty list.

1468 To support this query option, the MC shall supply the \$stop parameter in the CollectionTop field of the
1469 SupplyCustomRequestParameters (clause 12.2) request message. In the event that this query option is
1470 not supplied as part of the target URI for an HTTP/HTTPS GET operation, the MC shall supply a value of
1471 0xFFFF in this field; it shall not send the SupplyCustomRequestParameters just to supply a value of
1472 unlimited for the CollectionTop field.

1473 When processing an RDE read Operation for a resource collection, the RDE Device shall check the
1474 CollectionTop parameter from the SupplyCustomRequestParameters request message to determine the
1475 number of members to respond with, per [DSP0266](#). The RDE Device shall interpret a value of 0xFFFF as
1476 indicating that there is no limit to the number of members it should return for the referenced resource
1477 collection. In the event that the MC did not indicate the presence of extended request parameters, the
1478 RDE Device shall interpret this as a CollectionTop value of unlimited.

1479 **7.2.3.11.3 \$expand query option**

1480 The MC should support \$expand query options when provided as part of the target URI for a Redfish
1481 HTTP/HTTPS GET operation.

1482 The parameter for this query option is a string representing the links (Navigation properties) to expand in
1483 place, “gluing together” the results of multiple reads into a single JSON response payload. This parameter
1484 may be an absolute string specifying the exact link to be expanded, or it may be any of three wildcards.
1485 The first wildcard, an asterisk (*), means that all links should be expanded. The second wildcard, a dot (.),
1486 means that subordinate links (those that are directly referenced i.e., not in the Links Property section of
1487 the resource) should be expanded. The third wildcard, a tilde (~), means that dependent links (those that
1488 are not directly referenced i.e., in the Links Property section of the resource) should be expanded. See
1489 [DSP0266](#) for more details on the usage of \$expand.

1490 To support an expansion type wildcard received from the Redfish Client, the MC should send the PLDM-
1491 RDE-Expand-Type custom header described in clause 7.2.3.10.3.1 to the RDE Device via the
1492 SupplyCustomRequestParameters command of clause 12.2.

1493 If the \$levels query option qualifier is not present in conjunction with the \$expand query option, the MC
1494 shall treat this as equivalent to \$levels=1.

1495 To support the \$expand query option, the RDE Device should concatenate linked resource data into the
1496 BEJ data it returns for an RDE read Operation, using the bejResourceLinkExpansion PLDM data type
1497 described in Clause 5.3.23.

1498 7.2.3.11.4 \$levels query option qualifier

1499 The MC should support the \$levels qualifier to the \$expand query option when provided as part of the
1500 target URI for a Redfish HTTP/HTTPS GET operation or when provided implicitly by having \$expand
1501 provided as part of a Redfish HTTP/HTTPS GET operation without having the \$levels query option
1502 qualifier supplied.

1503 The parameter for this query option is an integer representing the number of schema links to expand into.
1504 If no \$level qualifier is present, the MC shall interpret this as equivalent to \$levels=1.

1505 To support this parameter, the MC can select between two choices: passing it on to the RDE Device or
1506 supporting it itself. The method by which this choice is made is implementation-specific and out of scope
1507 for this specification. If the RDE Device indicates that it cannot support \$levels expansion by setting the
1508 expand_support bit to zero in the DeviceCapabilitiesFlags in the response message to the
1509 NegotiateRedfishParameters command (clause 11.1), or if the expansion type is not “All Links” (see
1510 clause 7.2.3.11.3), the MC shall not select passing it to the RDE Device.

1511 If the MC chooses to pass this query option to the RDE Device, it shall transmit the supplied value to the
1512 RDE Device via the SupplyCustomRequestParameters command in the LinkExpand parameter.

1513 If the MC chooses to handle this query option itself, it shall recursively issue reads to “expand out” data
1514 for links embedded in data it reads. Such links may be identified during the BEJ decode process as tuples
1515 with a format of bejResourceLink (clause 5.3.21). The corresponding value of the node represents the
1516 Resource ID for the Redfish Resource PDR representing the data to embed within the structure of data
1517 already read. The \$levels qualifier dictates the depth of recursion for this process.

1518 When the RDE Device receives a LinkExpand value of greater than zero in extended request parameters
1519 as part of an RDE read operation, it shall “expand out” all resource links (as defined in [DSP0266](#)) to the
1520 indicated depth by encoding them as bejResourceLinkExpansions in the response BEJ data for the
1521 command. If the RDE Device previously did not set the expand_support flag in the
1522 DeviceCapabilitiesFlags field of the NegotiateRedfishParameters command, it may instead ignore the
1523 value (treating it as zero).

1524 Implementers should refer to [DSP0266](#) for more details and caveats to be applied when expanding links
1525 with \$levels > 1.

1526 7.2.3.11.5 \$select query option qualifier

1527 The MC may support \$select as a qualifier to the \$expand query option or as a standalone query option,
1528 provided in either case as part of the target URI for a Redfish HTTP/HTTPS GET operation.

1529 The parameter for this query option is a string containing a comma-separated list of properties to be
1530 retrieved from the GET operation; the caller is asking that all other properties be suppressed. See
1531 [DSP0266](#) for more details on the usage of \$select.

1532 If it supports this parameter, the MC should perform the GET operation normally up to the point of
1533 retrieving BEJ-formatted data from the RDE Device. When decoding the BEJ data, however, the MC
1534 should silently discard any property not part of the \$select list.

1535 No action is needed on the part of an RDE Device to support this query option.

1536 7.2.3.11.6 Excerpt query option

1537 The MC may support the excerpt query option when provided as part of a target URI for a Redfish
1538 HTTP/HTTPS GET operation. There is no parameter for this command.

1539 To support this parameter, the MC shall set the excerpt_flag in the OperationFlags field of the
1540 RDEOperationInit request command. Thereafter, no special treatment is required on the part of the MC.

1541 When the RDE Device is flagged that the client requested an excerpt, it may support the request by
1542 restricting properties returned in the read to those flagged with the excerpt schema annotation. If the
1543 schema does not contain any such flagged properties, or if the RDE Device does not support the excerpt
1544 query option, it shall return the complete resource.

1545 Further details of the excerpt query option may be found in [DSP0266](#).

1546 **7.2.3.12 HTTP/HTTPS status codes**

1547 The MC shall comply with [DSP0266](#) in all matters pertaining to the HTTP/HTTPS status codes returned
1548 for Redfish GET, PATCH, PUT, POST, DELETE, and HEAD operations. Typical status codes for
1549 operational errors may be found in clause 7.5.

1550 **7.2.3.13 Multihosting and Operations**

1551 A single RDE Device may find that it is attached to multiple MCs. This can introduce complications from
1552 concurrency if conflicting Operations are issued and requires an RDE Device to decide whether an
1553 Operation should be visible to an MC other than the one that issued it. Support for multiple MCs is out of
1554 scope for this specification. In particular, the behavior of the RDE Device in the face of concurrent
1555 commands from multiple MCs is undefined.

1556 **7.2.4 PLDM RDE Events**

1557 An Event is an abstract representation of any happening that transpires in the context of the RDE Device,
1558 particularly one that is outside of the normal command request/response sequence. A Redfish Message
1559 Event consists of JSON data that includes elements such as the index of a standardized text string and a
1560 collection of parameters that provide clarification of the specifics of the Event that has transpired. The full
1561 schema for Events may be found in the standard Redfish Message schema; additionally, OEM extensions
1562 to this schema are possible.

1563 In this specification, a second class of events, Task Executed Events, allow RDE Devices to report that a
1564 Task has finished executing and that the MC should retrieve Operation results. The data for these events
1565 includes elements such as the Operation identifier and the resource with which the Operation is
1566 associated.

1567 As with any other PLDM eventing, the RDE Device advertises that it supports Events by listing support for
1568 the PLDM for Platform Monitoring and Control SetEventReceiver command (see [DSP0248](#)). The MC, for
1569 its part, may then select between two methods by which it will know that Events are available. If the MC
1570 configured the RDE Device to use asynchronous events through the SetEventReceiver command, the
1571 RDE Device shall use the PLDM for Platform Monitoring and Control PlatformEventMessage command
1572 (see [DSP0248](#)) to inform the MC by sending the Event directly. Otherwise, the RDE Device can be
1573 configured to polling mode using the same SetEventReceiver command. The MC uses the PLDM for
1574 Platform Monitoring and Control PollForPlatformEventMessage command (see [DSP0248](#)) for this
1575 purpose. The selection of any polling interval is determined by the MC and is outside the scope of this
1576 specification.

1577 Whether retrieved synchronously or asynchronously, once the MC gets the Event, it may process it.
1578 Redfish Message Events are packaged using the redfishMessageEvent eventClass; Task Executed
1579 Events are packaged using the redfishTaskExecutedEvent eventClass (see [DSP0248](#) for both
1580 eventClasses).

1581 A PLDM Event Receiver may receive RDE events from the device before the RDE device registration and
1582 it can cache them or discard them based on Event Receiver capability. The PLDM Event Receiver (e.g.
1583 MC) can process the cached events after RDE Device registration is complete. The number of RDE
1584 events cached by the PLDM Event Receiver is outside the scope of this specification. If the MC wishes to
1585 guarantee processing of RDE events, it should defer the PLDM for Platform Monitoring and Control
1586 SetEventReceiver command until after the RDE device registration.

1587 Handling of Task Executed Events is described with Tasks in clause 7.2.5. For Redfish Message Events,
1588 the MC may decode the BEJ-formatted payload of Event data using the appropriate Event schema
1589 dictionary specific to the PDR from which the message was sent.

1590 For a more detailed view of the Event lifecycle, see clause 9.3.

1591 NOTE Events are optional in standard Redfish; however, support for Task Executed Events is mandatory in this
1592 specification if the RDE Device supports asynchronous execution for long-running Operations.

1593 7.2.4.1 [MC] Event subscriptions

1594 In Redfish, a client may request to be notified whenever a Redfish Event occurs. Per [DSP0266](#), to do so,
1595 the client uses a Redfish CREATE operation to add a record to the EventSubscription collection. This
1596 record in turn contains information on the various Event types that the client wishes to receive Events for.
1597 To unsubscribe, the client uses a Redfish DELETE operation to remove its record. Among other
1598 properties, the EventSubscription record contains a URI to which the Event should be forwarded. MCs
1599 that support Events shall support at least one Redfish event subscription.

1600 Event types are global across all schemas; there is no provision at this time ([DSP0266](#) v1.6) in Redfish
1601 for a client to subscribe to just one schema at a time. Further, there is generally no capacity for an RDE
1602 Device to send an HTTP/HTTPS record directly to an external recipient. Events are optional in Redfish;
1603 however, if the MC chooses to provide Event subscription support, it must comply with the following
1604 requirements:

- 1605 • The MC shall provide full support for the EventSubscription collection as a Redfish Provider per
1606 [DSP0266](#).
- 1607 • When it receives an Event subscription request (in the form of a Redfish CREATE operation on
1608 the EventSubscription collection), the MC shall parse the EventTypes array property of the
1609 request to identify the type or types of Events the client is interested in receiving
- 1610 • When the MC receives a Redfish Message Event from an RDE Device, it shall check the
1611 EventType of the Event received against the desired EventTypes for each active client. For
1612 each match, the MC shall forward the Event (translating any @Message.ExtendedInfo
1613 annotations, of course, from BEJ to JSON) to the client as a standard Redfish Provider for the
1614 Event service.

1615 7.2.5 Task support

1616 In PLDM for Redfish Device Enablement, every Redfish HTTP/HTTPS operation is effected as an RDE
1617 Operation. Most Operations, once sent to the RDE Device for execution, may be executed quickly and
1618 the results sent directly in the response message to the request message that triggered them.

1619 It may however transpire that in order for an RDE Device to complete an Operation, it requires more time
1620 than the available window within which the RDE Device is required to send a response. In this case, the
1621 RDE Device has two possible paths to follow. If the current number of extant Tasks is less than the RDE
1622 Device/MC capability intersection (as determined from the call to NegotiateRedfishParameters; see
1623 clause 11.1), the RDE Device shall mark the Operation as a long-running Task and execute it
1624 asynchronously. Otherwise, the RDE Device shall return ERROR_CANNOT_CREATE_OPERATION in
1625 its response message to indicate that no new Task slots are available (see clause 7.5).

1626 While the internal data structures used by an RDE Device to manage an Operation are outside the scope
1627 of this specification, they should include at a minimum the rdeOpID assigned (usually by the MC) when
1628 the Operation was first created. This allows the MC to reference the Task in subsequent commands to kill
1629 it (RDEOperationKill, clause 12.6) or query its status (RDEOperationStatus, clause 12.5).

1630 For its part, the MC shall provide full support for the Task collection as a Redfish Provider per [DSP0266](#).
1631 When the MC finds that an Operation has spawned a Task, it shall perform the following steps in order to
1632 comply with the requirements of [DSP0266](#):

- 1633 2) The MC shall instantiate a new TaskMonitor URL and a new member of the Task collection. The
1634 TaskMonitor URL should incorporate or reference (such as via a lookup table) the following data so
1635 that it can map from the TaskMonitor URL back to the correct Redfish resource – and thus the
1636 correct dictionary – for providing status query updates:
- 1637 a) The ResourceID for the resource to which the RDE Operation was targeted
 - 1638 b) The rdeOpID for the Operation itself
- 1639 2) The MC shall return response code 202, Accepted, to the client and include the Location response
1640 header populated with the TaskMonitor URL.
- 1641 3) In response to a subsequent Redfish GET Operation applied to the TaskMonitor URL or to the Task
1642 collection member, the MC shall invoke the RDEOperationStatus (see clause 12.5) command to
1643 obtain the latest status for the Operation and communicate it to the client in accordance with
1644 [DSP0266](#). If the GET was applied to a TaskMonitor URL and the Operation has been completed, the
1645 MC shall supply the completed results to the client.
- 1646 a) If the result of the RDEOperationStatus command was that the Operation has finished
1647 execution, the MC shall delete both the TaskMonitor URL and the Task collection member
1648 associated with the Operation.
- 1649 4) In response to a Redfish DELETE Operation applied to the TaskMonitor URL or to the Task
1650 collection member, the MC shall attempt to abort the associated Operation via the RDEOperationKill
1651 (see clause 12.6) command. It shall then remove both the TaskMonitor URL and the Task collection
1652 member.
- 1653 5) If the RDE Operation finishes before the client polls the TaskMonitor URL, the MC may collect and
1654 store the results of the Operation.
- 1655 a) In accordance with [DSP0266](#), the MC should retain Operation results until the client retrieves
1656 them. It may refuse to accept further Operations until previous results have been claimed.
 - 1657 b) If the client attempts to collect Operation results after the MC has discarded them, the MC shall
1658 respond with an error HTTP status code as defined in [DSP0266](#).

1659 When the RDE Device finishes execution of a Task, it generates a Task Executed Event to inform the MC
1660 of this status change. The MC can then retrieve the results (via RDEOperationStatus) and eventually
1661 forward them to the client. To mark the Task as complete and allow the RDE Device to discard any
1662 internal data structures used to manage the Task, the MC shall call RDEOperationComplete (clause
1663 12.4).

1664 For a more detailed overview of the Operation/Task lifecycle from the MC's perspective, see clause
1665 7.2.3.9.1.3. A detailed flowchart of the Operation/Task lifecycle may be found in clause 9.2.2, and a finite
1666 state machine for the Task lifecycle (from the RDE Device's perspective) may be found in clause 9.2.3.

1667 7.3 Type code

1668 Refer to [DSP0245](#) for a list of PLDM Type Codes in use. This specification uses the PLDM Type Code
1669 000110b as defined in [DSP0245](#).

1670 7.4 Transport protocol type supported

1671 PLDM can support bindings over multiple interfaces; refer to [DSP0245](#) for the complete list. All transport
1672 protocol types can be supported for the commands defined in Table 51.

1673 7.5 Error completion codes

1674 Table 37 lists PLDM completion codes for Redfish Device Enablement. The usage of individual error
1675 completion codes is defined within each of the PLDM command clauses. When communicating results
1676 back to the client, implementations should provide HTTP error codes as described below.

Table 37 – PLDM for Redfish Device Enablement completion codes

Value	Name	Description	HTTP Error Code
Various	PLDM_BASE_CODES	Refer to DSP0240 for a full list of PLDM Base Code Completion values that are supported.	See below.
128 (0x80)	ERROR_BAD_CHECKSUM	A transfer failed due to a bad checksum and should be restarted.	MC should retry transfer. If retry fails, 500 Internal Server Error
129 (0x81)	ERROR_CANNOT_CREATE_OPERATION	An Operation-based command failed because the RDE Device could not instantiate another Operation at this time.	503 Service Unavailable
130 (0x82)	ERROR_NOT_ALLOWED	The client and/or MC is not allowed to perform the requested Operation.	405 Method Not Allowed
131 (0x83)	ERROR_WRONG_LOCATION_TYPE	A Create, Delete, or Action Operation attempted against a location that does not correspond to the right type.	405 Method Not Allowed
132 (0x84)	ERROR_OPERATION_ABANDONED	An Operation-based command other than completion was attempted with an Operation that has timed out waiting for the MC to progress it in the Operation lifecycle.	410 Gone
133 (0x85)	ERROR_OPERATION_UNKILLABLE	An attempt was made to kill an Operation that has already finished execution or that cannot be aborted.	409 Conflict
134 (0x86)	ERROR_OPERATION_EXISTS	An Operation initialization was attempted with an rdeOpID that is currently active.	N/A – MC retries with a new rdeOpID
135 (0x87)	ERROR_OPERATION_FAILED	An Operation-based command other than completion was attempted with an Operation that has encountered an error in the Operation lifecycle.	400 Bad Request
136 (0x88)	ERROR_UNEXPECTED	A command was sent out of context, such as sending SupplyCustomRequestParameters when Operation initialization flags did not indicate that the Operation requires them.	500 Internal Server Error
138 (0x89)	ERROR_UNSUPPORTED	An attempt was made to initialize an operation not supported by the RDE Device, to write to a property that the RDE Device does not support, or a command was issued containing a text string in a format that the recipient cannot interpret.	400 Bad Request
144 (0x90)	ERROR_UNRECOGNIZED_CUSTOM_HEADER	The RDE Device received a custom PLDM-RDE header (via SupplyCustomRequestParameters) that it does not support.	412 Precondition Failed

Value	Name	Description	HTTP Error Code
145 (0x91)	ERROR_ETAG_MATCH	The RDE Device received one or more ETags that did not match an If-Match or If-None-Match request header.	412, Precondition Failed (If-Match) or 304, not modified (If-None-Match)
146 (0x92)	ERROR_NO_SUCH_RESOURCE	An Operation command was invoked with a resource ID that does not exist.	404, Not Found
147 (0x93)	ETAG_CALCULATION_ONGOING	Calculating the ETag in response to the GetResourceETag command is taking too long to provide an immediate response.	N/A – MC retries with the same command later
148 (0x94)	ERROR_INSUFFICIENT_STORAGE	The RDE Device lacks storage to process the request, such as being unable to return a payload due to its size.	507 Insufficient Storage

1678 HTTP Error codes returned when Operations complete with standard PLDM completion codes should be
 1679 as follows:

1680 **Table 38 – HTTP codes for standard PLDM completion codes**

Name	Description	HTTP Error Code
SUCCESS	Normal success	200 Success, 202 Accepted for an Operation that spawned a Task, or 204 No Content for an Action that has no response
ERROR	Generic error	400 Bad Request
ERROR_INVALID_DATA	Invalid data or a bad parameter value	500 Internal Server Error
ERROR_INVALID_LENGTH	Incorrectly formatted request method	500 Internal Server Error
ERROR_NOT_READY	Device transiently busy	503 Service Unavailable
ERROR_UNSUPPORTED_PLDM_CMD	Command not supported	501 Not Implemented
ERROR_INVALID_PLDM_TYPE	Not a supported PLDM type	501 Not Implemented

1681 **7.6 Timing specification**

1682 Table 39 below defines timing values that are specific to this document. The table below defines the
 1683 timing parameters defined for the PLDM Redfish Specification. In addition, all timing parameters listed in
 1684 [DSP0240](#) for command timeouts, command response times, and number of retries shall also be followed.

1685

Table 39 – Timing specification

Timing specification	Symbol	Min	Max	Description
PLDM Base Timing	PN _x PT _x (see DSP0240)	(See DSP0240)	(See DSP0240)	Refer to DSP0240 for the details on these timing values.
Operation/Transfer abandonment	T _{abandon}	120 seconds	none	Time between when the RDE Device is ready to advance an Operation through the Operation lifecycle and when the MC must have initiated the next step. If the MC fails to do so, the RDE Device may consider the Operation as abandoned. Also used in follow up to a GetSchemaDictionary command to mark the time between when the MC receives one chunk of dictionary data and when it must request the next chunk. If the MC fails to do so, the RDE Device may consider the transfer as abandoned.

1686 **8 Binary Encoded JSON (BEJ)**

1687 This clause defines a binary encoding of Redfish JSON data that will be used for communicating with
 1688 RDE Devices. At its core, BEJ is a self-describing binary format for hierarchical data that is designed to
 1689 be straightforward for both encoding and decoding. Unlike in ASN.1, BEJ uses no contextual encodings;
 1690 everything is explicit and direct. While this requires the insertion of a bit more metadata into BEJ encoded
 1691 data, the tradeoff benefit is that no lookahead is required in the decoding process. The result is a
 1692 significantly streamlined representation that fits in a very small memory footprint suitable for modern
 1693 embedded processors.

1694 **8.1 BEJ design principles**

1695 The core design principles for BEJ are focused around it being a compact binary representation of JSON
 1696 that is easy for low-power embedded processors to encode, decode, and manipulate. This is important
 1697 because these ASICs typically have highly limited memory and power budgets; they must be able to
 1698 process data quickly and efficiently. Naturally, it must be possible to fully reconstruct a textual JSON
 1699 message from its BEJ encoding.

1700 The following design principles guided the development of BEJ:

- 1701 1) It must be possible to support full expressive range of JSON.
- 1702 2) The encoding should be binary and compact, with as much of the encoding as possible
 1703 dedicated to the JSON data elements. The amount of space afforded to metadata that conveys
 1704 elements such as type format and hierarchy information should be carefully limited.
- 1705 2) There is no need to support multiple encoding techniques for one type of data; there is therefore
 1706 no need to distinguish which encoding technique is in use.
- 1707 3) Schema information – such as the names of data items – does not need to be encoded into BEJ
 1708 because the recipient can use a prior knowledge of the data organization to determine semantic

1709 information about the encoded data. In contrast to JSON, which is unordered, BEJ must adopt
 1710 an explicit ordering for its data to support this goal.

1711 4) The need for contextual awareness should be minimized in the encoding and decoding process.
 1712 Supporting context requires extra lookup tables (read: more memory) and delays processing
 1713 time. Everything should be immediately present and directly decodable. Giving up a few bytes
 1714 of compactness in support of this goal is a worthwhile tradeoff.

1715 **8.2 SFLV tuples**

1716 Each piece of JSON data is encoded as a tuple of PLDM type bejTuple and consists of the following:

- 1717 1) Sequence number: the index within the canonical schema at the current hierarchy level for the
 1718 datum. For collections and arrays, the sequence number is the 0-based array index of the
 1719 current element.
- 1720 2) Format: the type of data that is encoded.
- 1721 3) Length: the length in bytes of the data.
- 1722 4) Value: the actual data, encoded in a format-specific manner.

1723 These tuple elements collectively describe a single piece of JSON data; each piece of JSON data is
 1724 described by a separate tuple. Requirements for each tuple element are detailed in the following clauses.

1725 SFLV tuples are represented by elements of the bejTuple PLDM type defined in clause 5.3.5.

1726 **8.2.1 Sequence number**

1727 The Sequence Number tuple field serves as a stand-in for the JSON property name assigned to the data
 1728 element the tuple encodes. Sequence numbers align to name strings contained within the dictionary for a
 1729 given schema. Sequence numbers are represented by elements of the bejTupleS PLDM type defined in
 1730 clause 5.3.6.

1731 The low-order bit of a sequence number shall indicate the dictionary to which it belongs according to the
 1732 following table:

1733 **Table 40 – Sequence number dictionary indication**

Bit Pattern	Dictionary
0b	Main Schema Dictionary (as was defined in the bejEncoding PLDM object for this tuple)
1b	Annotation Dictionary

1734 **8.2.2 Format**

1735 The Format tuple field specifies the kind of data element that the tuple is representing.

1736 Formats are represented by elements of the bejTupleF PLDM type defined in clause 5.3.7.

1737 **8.2.3 Length**

1738 The Length tuple field details the length in bytes of the contents of the Value tuple field.

1739 Lengths are represented by elements of the bejTupleL PLDM type defined in clause 5.3.8.

1740 **8.2.4 Value**

1741 The Value tuple field contains an encoding of the actual data value for the JSON element described by
 1742 this tuple. The format of the value tuple field is variable but follows directly from the format code in the
 1743 Format tuple field.

1744 The following JSON data types are supported in BEJ:

1745 **Table 41 – JSON data types supported in BEJ**

BEJ Type	JSON Type	Description
Null	null	An empty data type
Integer	number	A whole number: any element of JSON type number that contains neither a decimal point nor an exponent
Enum	enum	An enumeration of permissible values in string format
String	string	A null-terminated UTF-8 text string
Real	number	A non-whole number: any element of JSON type number that contains at least one of a decimal point or an exponent
Boolean	boolean	Logical true/false
Bytestring	string (of base-64 encoded data)	Binary data
Set	No named type; data enclosed in { }	A named collection of data elements that may have differing types
Array	No named type; data enclosed in []	A named collection of zero or more copies of data elements of a common type
Choice	special	The ability of a named data element to be of multiple types
Property Annotation	special	An annotation targeted to a specific property, in the format property@annotation
Unrecognized	special	Used to perform a pass-through encoding of a data element for which the name cannot be found in a dictionary for the corresponding schema
Schema Link	special	Used to capture JSON references to external schemas
Expanded Schema Link	special	Used to expand data from a linked external schema

1746 If the deferred_binding flag (see the bejTupleF PLDM type definition in clause 5.3.7) is set, the string
 1747 encoded in the value tuple element contains substitution macros that the MC is to supply on behalf of the
 1748 RDE Device when populating a message to send back to the client. See clause 8.3 for more details.

1749 Values are represented by elements of the bejTupleV PLDM type defined in clause 5.3.9.

1750 **8.3 Deferred binding of data**

1751 The data returned to a client from a Redfish operation typically contains annotation metadata that specify
 1752 URIs and other bits of information that are assigned by the MC when it performs RDE Device discovery
 1753 and registration. In practice, the only way for an RDE Device to know the values for these annotations
 1754 would be for it to somehow query the MC about them. Instead, we define substitution macros that the

1755 RDE Device may use to ask the MC to supply these bits of information on its behalf. RDE Devices shall
 1756 not invoke substitution macros for information that they know and can provide themselves.

1757 All substitution macros are bracketed with the percent sign (%) character. While it would in theory be
 1758 possible for the MC to check every string it decodes for the presence of this escape character, in practice
 1759 that would be an inefficient waste of MC processing time. Instead, the RDE Device shall flag any string
 1760 containing substitution macros with the deferred binding bit set to inform the MC of their presence; the
 1761 MC shall only perform macro substitution if the deferred binding bit is set. The MC shall support the
 1762 deferred bindings listed in Table 42.

1763 **Table 42 – BEJ deferred binding substitution parameters**

Macro	Data to be substituted	Example substitutions
%%	A single % character	%
%L<resource-ID>	The MC-assigned URI of an RDE Provider defined resource (specified by a resource ID within the target PDR), or /invalid.PDR<resource-ID> if unrecognized resource ID	/invalid.PDR123
%P<resource-ID>.PAGE<pagination-offset>	The MC-assigned URI of an RDE Provider defined resource (specified by a resource ID within the target PDR) with a given numerical pagination offset, or /invalid.PDR<resource-ID>.PAGE<pagination-offset> if unrecognized resource ID or pagination offset < 1	/invalid.PDR101.PAGE-1
%PD	The MC-assigned URI for an MC-managed PCIeDevice.PCIeDevice correlating to this RDE Device, or /invalid.PCIeDevice if one cannot be identified. If the RDE Device manages its own PCIeDevice.PCIeDevice resource, it shall use the %L binding when referring to it,	/invalid.PCIeDevice
%PF<function-info>	The MC-assigned URI for an MC-managed PCIeFunction.PCIeFunction correlating to the RDE Device’s function matching function-info, or /invalid.PCIeFunction.<function-info> if one cannot be identified. Function-info shall be a string of lower-case hexadecimal digits corresponding to the PCIe function number for the function. If the RDE Device manages its own PCIeFunction.PCIeFunction resource, it shall use the %L binding when referring to it,	/invalid.PCIeFunction.nonhexdigits /redfish/v1/chassis/1/PCIeDevices/NIC/PCIeFunctions/1

Macro	Data to be substituted	Example substitutions
%PI	The MC-assigned URI for an MC-managed PCIeDevice.PCIeInterface correlating to this RDE Device, or /invalid.PCIeInterface if one cannot be identified. If the RDE Device manages its own PCIeDevice.PCIeInterface resource, it shall use the %L binding when referring to it	/invalid.PCIeInterface
%S	The MC-assigned link to the ComputerSystem resource within which the RDE Device is located	/redfish/v1/Systems/437XR1138R2
%C	The MC-assigned link to the Chassis resource within which the RDE Device is located	/redfish/v1/Chassis/1U
%M	The metadata URL for the service	/redfish/v1/\$metadata
%T<resource-ID>.<n>	The MC-assigned target URI for the n th Action from the Redfish Action PDR or PDRs linked to a resource within a Redfish Resource PDR, or “/invalid.<resource-ID>.<n>” if no such action exists	/redfish/v1/Systems/437XR1138R2/Storage/1/Actions/Storage.SetEncryptionKey /invalid.123.6
%I<resource-ID>	The MC-assigned instance identifier for the collection element representing an RDE Device (specified by the resource ID of the target PDR), or “invalid” if the PDR does not correspond to a resource immediately contained within a collection managed by the MC	437XR1138R2 invalid
%U	The UEFI Device Path assigned to the RDE Device by the MC and/or BIOS	PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x0)/Scsi(0xA, 0x0)
%.	Terminates a previous substitution. Shall be used only in the event that numeric data immediately follows a %T, %P, or %L macro	n/a
%B	The MC-assigned URI for an MC-managed Battery.Battery correlating to the RDE Device or /invalid.Battery if one cannot be identified. If the RDE Device manages its own Battery resource, it shall use the %L binding when referring to it,	/invalid.Battery.nonhexdigits /redfish/v1/chassis/1/PowerSubsystem/Batteries/1
Any other character preceded by a % character	None – the MC shall pass the sequence exactly as found	%p %X

1764 **8.4 BEJ encoding**

1765 This clause presents implementation considerations for the BEJ encoding process. For standard resource
 1766 encoding (as opposed to annotations), the BEJ conversion dictionary is built to encode the same
 1767 hierarchical data format as the schema itself. Implementations should therefore track their context inside
 1768 the dictionary in parallel with tracking their location in the data to be encoded. While not mandatory, a
 1769 recursive implementation will prove in most cases to be the easiest approach to realize this tracking.

1770 Like with JSON encodings of data, there is no defined ordering for properties in BEJ data; encoders are
1771 therefore free to encode properties in any order.

1772 8.4.1 Conversion of JSON data types to BEJ

1773 Recognition of [JSON](#) data types enables them to be encoded properly. In Redfish, every property is
1774 encoded in the format "property_name" : property_value. Whitespace between syntactic elements is
1775 ignored in JSON encodings.

1776 8.4.1.1 JSON objects

1777 A JSON object consists of an opening curly brace ('{'), zero or more comma-separated properties, and
1778 then a closing curly brace ('}'). JSON objects shall be encoded as BEJ sets with the properties inside the
1779 curly braces encoded recursively as the value tuple contents of the BEJ set. Following the precedent
1780 established in JSON, the properties contained within a JSON object may be encoded in BEJ in any order.
1781 In particular, the encoding order for a collection of properties is not required to match their respective
1782 sequence numbers.

1783 8.4.1.2 JSON arrays

1784 A JSON array consists of an opening square brace ('['), zero or more comma-separated JSON values all
1785 of a common data type (typically objects in Redfish), and then a closing square brace. JSON arrays shall
1786 be encoded as BEJ arrays with the data inside the square braces encoded recursively as instances of the
1787 value tuple contents of the BEJ array. The immediate contents of a JSON array shall be encoded in order
1788 corresponding to their array indices.

1789 The sequence numbers for BEJ array immediate child elements shall match the zero-based array index
1790 of the children. These sequence numbers are not represented in the dictionary; it is the responsibility of a
1791 BEJ encoder/decoder to understand that this is how array data instances are handled.

1792 8.4.1.3 JSON numbers

1793 In JSON, there is no distinction between integer and real data; both are collected together as the number
1794 type. For BEJ, numeric data shall be encoded as a BEJ integer if it contains neither a decimal point nor
1795 an exponentiation marker ('e' or 'E') and as a BEJ real otherwise.

1796 8.4.1.4 JSON strings

1797 When converting JSON strings to BEJ format, a null terminator shall be appended to the string.

1798 8.4.1.5 JSON Boolean

1799 In JSON, Boolean data consists of one of the two sentinels "true" or "false". These sentinels shall be
1800 encoded as BEJ Boolean data with an appropriate value field.

1801 8.4.1.6 JSON null

1802 In JSON, null data consists of the sentinel "null". This sentinel shall be encoded as BEJ Null data only if
1803 the datatype for the property in the schema is null. For a nullable property (identified via the third tag bit
1804 from the dictionary entry or by the schema), null data shall be encoded as its standard type (from the
1805 dictionary) with length zero and no value tuple element.

1806 8.4.2 Resource links

1807 Most Redfish schemas contain links to other schemas within their properties, formatted as @odata.id
1808 annotations. When encoding these links in BEJ, the URI may be encoded as any of bejString,

1809 bejResourceLink, or bejResourceLinkExpansion. If encoded as a bejString, deferred binding substitutions
1810 may be employed as needed to complete the reference.

1811 When encoding a BEJ payload as part of an RDE create, update or action operation, the MC should use
1812 the following criteria when encoding resource links:

- 1813 1. If the resource link can be mapped to a Redfish resource PDR, the MC should encode the link
1814 using the bejResourceLink data type.
- 1815 2. If the resource link cannot be mapped to a Redfish resource PDR, and is a well-formed link, the
1816 MC should use a deferred binding bejString to encode the link.
- 1817 3. For a malformed link, the MC should reject the operation using a HTTP status code of 400 Bad
1818 Request.

1819 8.4.3 Registry items

1820 Redfish messages contain items from collated collections called registries. When encoding Redfish
1821 message registries in BEJ, the string may be encoded as either bejString or bejRegistryItem.

1822 8.4.4 Annotations

1823 Redfish annotations may be recognized as properties with a name string containing the “at” sign (“@”).
1824 Several annotations are defined in Redfish, including some that are mandatory for inclusion with any
1825 Redfish GET Operation. The RDE Device is responsible for ensuring that these mandatory annotations
1826 are included in the results of an RDE read Operation.

1827 Annotations in Redfish have two forms:

- 1828 • Standalone form annotations have the form “@annotation_class.annotation_name” :
1829 annotation_value.
 - 1830 – Example: “@odata.id”: “/redfish/v1/Systems/1/”
 - 1831 – Standalone annotations shall be encoded with the BEJ data type listed in the annotation
1832 dictionary in the row matching the annotation name string
- 1833 • Property annotation form annotations have the form
1834 “property@annotation_class.annotation_name” : annotation_value.
 - 1835 – Example: “ResetType@Redfish.AllowableValues” : [“On”, “PushPowerButton”]
 - 1836 – Property annotation form annotations shall be encoded with the BEJ Property Annotation
1837 data type; the annotation value shall be encoded as a dependent child of the annotation
1838 entry. See clause 5.3.20.

1839 NOTE Unlike major schema resource properties, annotations have a flat namespace from which sequence numbers
1840 are drawn. To identify the sequence number for an annotation, an encoder should start at the root of the annotation
1841 dictionary and then find the string matching the annotation name (including the ‘@’ sign and the annotation source)
1842 within this set. In particular, the sequence number for an annotation is independent of the current encoding context.

1843 Special handling is required when the RDE Device sends a message annotation to the MC. The related
1844 properties property inside the annotation’s data structure is formatted as an array of strings, but the RDE
1845 Device has only sequence numbers to work with: the RDE Device may not be able to supply the property
1846 name for the sequence number. If the RDE Device knows the name of the related property that is
1847 relevant for the message annotation, it may supply the name directly as an array element. Otherwise, it
1848 shall encode into the array element a BEJ locator by concatenating the following string components:

1849

Table 43 – Message annotation related property BEJ locator encoding

Description
Delimiter Shall be ':'
ComponentCount The number N of sequence numbers in the fields below, stringified
Delimiter Shall be ':'
Locator Component [0] Sequence number [0], stringified
Delimiter Shall be ':'
Locator Component [1] Sequence number [1], stringified
Delimiter Shall be ':'
Locator Component [2] Sequence number [2], stringified
Delimiter Shall be ':'
...
Delimiter Shall be ':'
Locator Component [N – 1] Sequence number [N – 1], stringified

1850 **8.4.4.1 Nested Annotations**

1851 The data format for an annotation may be a simple property such as a string or an integer, but it may also
 1852 be a compound property such as a set. In this latter case it is further possible that the set can itself
 1853 contain an annotation. To distinguish the case where the sequence number for annotation data refers
 1854 anew to a top-level annotation instead of to a property within the set for an annotation, the format byte of
 1855 the BEJ tuple for that annotation shall have the read_only_property_and_top_level_annotation bit set to
 1856 1b. When encoding nested annotations, the BEJ encoding version shall be set to 1.1.0 (0xF1F1F000).

1857 **8.4.5 Choice encoding for properties that support multiple data types**

1858 If the encoder finds a property that is listed in the dictionary as being of type BEJ choice, it shall encode
 1859 the property with type bejChoice in the BEJ format tuple element. The actual value and selected data type
 1860 shall be encoded as a dependent child of the tuple containing the bejChoice element. See clauses 5.3.19
 1861 and 7.2.3.3.

1862 **8.4.6 Properties with invalid values**

1863 If the MC is encoding an update request from a client that includes a property value that does not match a
 1864 required data type according to the dictionary it is translating from, the MC shall in accordance with the
 1865 Redfish standard [DSP0266](#) respond to the client with HTTP status code 400 and a
 1866 @Message.ExtendedInfo annotation specifying the property with the value format error (see

1867 PropertyValueFormatError, PropertyValueTypeError in the Redfish base message registry). Similarly, if
 1868 the value supplied for a property such as an enumeration does not match any required values, the MC
 1869 shall in accordance with the Redfish standard [DSP0266](#) respond to the client with HTTP status code 400
 1870 and a @Message.ExtendedInfo annotation specifying the property with a value not in the accepted list
 1871 (see PropertyValueNotInList in the Redfish base message registry). In either case, the MC shall not
 1872 initiate an RDE Operation corresponding to the client request. Properties missing from dictionaries

1873 When encoding JSON data, an encoder may find that the name of a property does not correspond to a
 1874 string found in the dictionary. If the encoder is the RDE Device, this should never happen as the RDE
 1875 Device is responsible for the dictionary. This situation therefore represents a non-compliant RDE
 1876 implementation.

1877 If the MC finds that a property does not correspond to a string found in the dictionary from an RDE
 1878 Device, it should in accordance with the Redfish standard [DSP0266](#) respond to the client with HTTP
 1879 status code 200 or 400 and an annotation specifying the property as unsupported (see PropertyUnknown
 1880 in the Redfish base message registry). The MC may continue to process the client request, omitting the
 1881 unrecognized property or properties.

1882 Any attempt to write to an action, or to the parameters or return type within an action set, shall be
 1883 processed by the MC in the same manner as described in this clause for missing properties for write or
 1884 update operations; naturally, this does not apply to supplying parameters for action operations.

1885 Any attempt to write to an action parameter outside of an Action RDE Operation shall be processed as
 1886 described in this clause, as if the parameter were not present in the dictionary.

1887 **8.5 BEJ decoding**

1888 This clause presents implementation considerations for the BEJ decoding process.

1889 Properties in BEJ data may be encoded in any order. Decoders must therefore be prepared to accept
 1890 data in whatever order it was encoded.

1891 **8.5.1 Conversion of BEJ data types to JSON**

1892 When decoding from BEJ to JSON, the following rules shall be followed. In each of the following,
 1893 “property_name” shall be taken to mean the name of the property or annotation as decoded from the
 1894 relevant dictionary. For all data types, if the length tuple field is zero, the data shall be decoded as
 1895 follows:

1896 “property_name” : null

1897 When multiple properties appear sequentially within a set, they shall be delimited with commas.

1898 **8.5.1.1 BEJ Set**

1899 A BEJ Set shall be decoded to the following format, with the text inside angle brackets (‘<’, ‘>’) replaced as
 1900 indicated:

1901 “property_name” : { <set dependent children decoded individually as a comma-separated list> }

1902 **8.5.1.2 BEJ Array**

1903 A BEJ Array shall be decoded to the following format, with the text inside angle brackets (‘<’, ‘>’) replaced
 1904 as indicated:

1905 “property_name” : [<array dependent children decoded individually as a comma-separated list>]

1906 8.5.1.3 BEJ Integer and BEJ Real

1907 BEJ Integers and BEJ Reals shall be decoded to the following format, with the text inside angle brackets
1908 ('<', '>') replaced as indicated:

1909 "property_name" : "<decoded numeric value>"

1910 8.5.1.4 BEJ String

1911 BEJ Strings shall be decoded to the following format, with the text inside angle brackets ('<', '>') replaced
1912 as indicated. When converting BEJ strings to JSON format, the null terminator shall be dropped as JSON
1913 string encodings do not include null terminators.

1914 "property_name" : "<decoded string value>"

1915 8.5.1.5 BEJ Boolean

1916 BEJ Booleans shall be decoded to the following format, with the text inside angle brackets ('<', '>')
1917 replaced as indicated (note that the "true" and "false" sentinels are not encased in quote marks):

1918 "property_name" : < or , depending on the decoded value>

1919 8.5.1.6 BEJ Null

1920 BEJ Null shall be decoded to the following format:

1921 "property_name" :

1922 8.5.1.7 BEJ Resource Link

1923 A BEJ Resource Link shall be decoded to the following format, with the text inside angle brackets ('<', '>')
1924 replaced as indicated.

1925 "property_name" : "<URI for the resource corresponding the Redfish Resource PDR with the
1926 supplied ResourceID>"

1927 MCs shall be aware that either a BEJ Resource Link or a BEJ Resource Link Expansion may be encoded
1928 for a dictionary entry that lists its type as BEJ Resource Link.

1929 8.5.1.8 BEJ Resource Link expansion

1930 A BEJ Resource Link Expansion shall be decoded to the following format, with the text inside angle
1931 brackets ('<', '>') replaced as indicated.

1932 <full resource data for the Redfish Resource PDR corresponding to the supplied ResourceID>

1933 NOTE property_name is not included in the decoded JSON output in this case.

1934 If the supplied ResourceID is zero and the parent resource is a collection, the MC shall use the
1935 COLLECTION_MEMBER_TYPE schema dictionary obtained from the collection resource (rather than
1936 trying to use a dictionary from the members) to decode resource data.

1937 MCs shall be aware that either a BEJ Resource Link or a BEJ Resource Link Expansion may be encoded
1938 for a dictionary entry that lists its type as BEJ Resource Link.

1939 8.5.2 Annotations

1940 This clause documents the approach for decoding the two types of Redfish annotations to JSON text.

1941 8.5.2.1 Standalone annotations

1942 Standalone annotations (data from decoded from the annotation dictionary) shall be decoded to the
1943 following format, with the bit inside angle brackets ('<', '>') replaced as indicated:

1944 "@annotation_class.annotation_name" : "<decoded annotation value>"

1945 8.5.2.2 BEJ property annotations

1946 BEJ Property Annotations shall be decoded to the following format, with the bit inside angle brackets ('<',
1947 '>') replaced as indicated:

1948 "property_name@annotation_class.annotation_name" : "<decoded annotation value from the
1949 annotation's dependent child node>"

1950 8.5.2.3 [MC] Related Properties in message annotations

1951 When a message annotation is sent from the RDE Device to the MC, the related properties field of
1952 message annotations requires special handling in RDE. Specifically, the array element string values are
1953 BEJ locators to individual properties, may be encoded as a colon-delimited string (see clause 8.4.3).
1954 When decoding, the MC shall check the first character of the supplied string. If it is a colon (:), the MC
1955 shall extract the individual sequence numbers for the BEJ locator, and then use them to identify the
1956 property name to send back to the client for the annotation. If the first character of the supplied string is
1957 not a colon, the MC shall return the supplied string unmodified.

1958 8.5.3 Sequence numbers missing from dictionaries

1959 It may transpire that when decoding BEJ data, a decoder finds a sequence number not in its dictionary.
1960 The handling of this case differs between the RDE Device and the MC.

1961 If the RDE Device finds an unrecognized sequence number as part of the payload for a put, patch, or
1962 create operation, the RDE Device shall in accordance with the Redfish standard [DSP0266](#) respond with
1963 an annotation specifying the sequence number as an unsupported property (see PropertyUnknown in the
1964 Redfish base message registry). The RDE Device may continue to decode the remainder of the payload
1965 and perform the requested Operation upon the portion it understands.

1966 If the MC finds an unrecognized sequence number as part of the response payload for a get or action
1967 Operation, or as part of a @Message.ExtendedInfo annotation response for any other Operation, it shall
1968 treat this as a failure on the part of the RDE Device and respond to the client with HTTP status code 500,
1969 Internal Server Error.

1970 8.5.4 Sequence numbers for read-only properties in modification Operations

1971 If the RDE Device is performing a modification operation (create, put, patch, or some actions), and it finds
1972 a sequence number corresponding to a property that is read-only, the RDE Device should in accordance
1973 with the Redfish standard [DSP0266](#) respond with an annotation specifying the sequence number as a
1974 non-updateable property (see PropertyNotWritable in the Redfish base message registry). The RDE
1975 Device may continue to decode and update with the remainder of the payload.

1976 8.5.5 Annotations for RDE Devices

1977 RDE Devices shall conditionally implement support for the following standard Redfish annotations
1978 according to the following table:

1979

Table 44 – Conditionally Required Annotations for RDE Devices

Annotation	Condition when required
PropertyNotWritable	RDE Device supports RDE Write Operations and Write Operation attempts to modify a read-only property.
PropertyUnknown	Operation contains a sequence number that is unrecognized or unsupported.
PropertyValueError	RDE Device supports RDE Write Operations and Write Operation attempts to modify a read/write property to an illegal or unsupported value.
ActionParameterValueError	RDE Device supports RDE Action Operations and RDE Action Operation contains a parameter with an illegal or unsupported value.
QueryParameterValueError	RDE Device supports read queries, but Head or Read Operation contains a query parameter with an illegal or unsupported value.
QueryNotSupported	RDE Device supports read queries, but does not support a query operation contained in the requested Head or Read Operation.

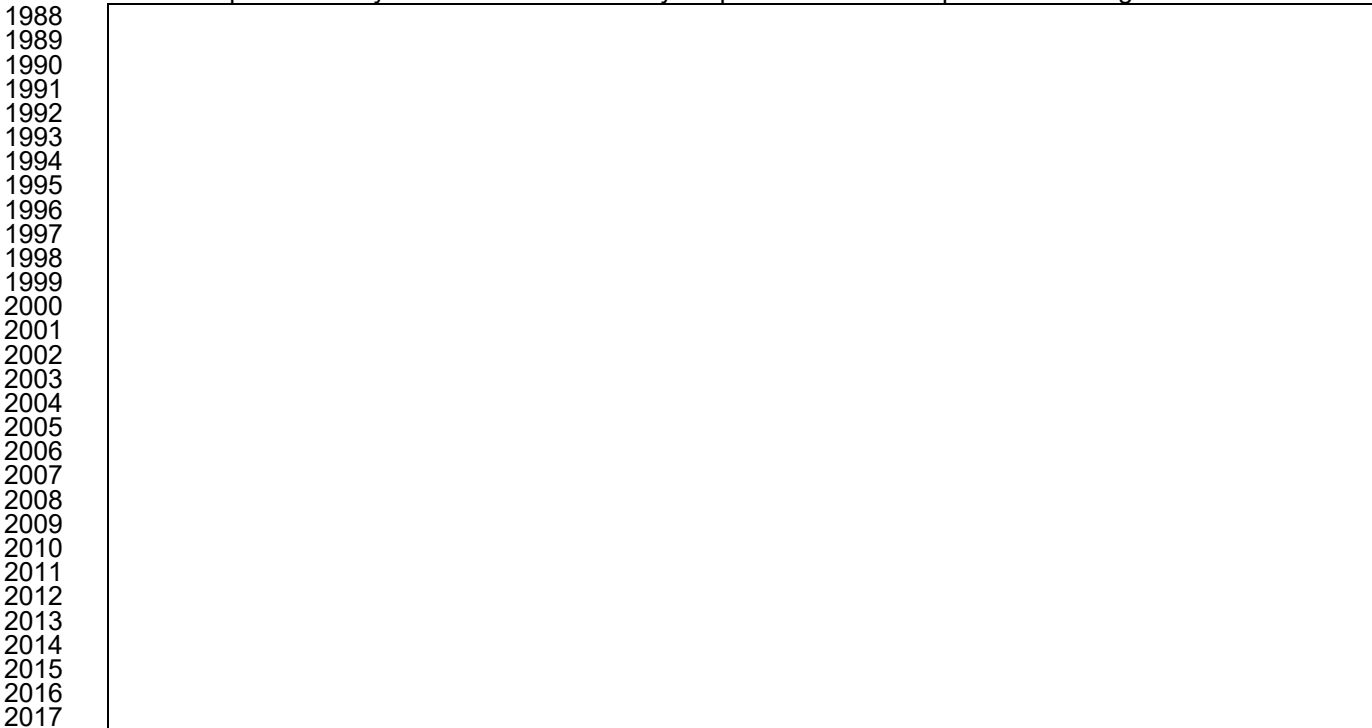
1980 **8.6 Example encoding and decoding**

1981 The following examples demonstrate the BEJ encoding and decoding processes. For illustrative
 1982 purposes, we show the data collected in an XML form that happens to align with the schema; however,
 1983 there is no requirement that data be stored in this form. Indeed, it is very unlikely that any RDE Device
 1984 would do so.

1985 The examples in this clause use the example dictionary from clause 8.6.1.

1986 **8.6.1 Example dictionary**

1987 The example dictionary is based on the DummySimple JSON schema presented in Figure 6:



1988
 1989
 1990
 1991
 1992
 1993
 1994
 1995
 1996
 1997
 1998
 1999
 2000
 2001
 2002
 2003
 2004
 2005
 2006
 2007
 2008
 2009
 2010
 2011
 2012
 2013
 2014
 2015
 2016
 2017

2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073



2074

Figure 6 – DummySimple schema

2075

NOTE This is not a published DMTF Redfish schema.

2076

In tabular form, the dictionary for DummySimple appears as shown in Table 45:

2077

Table 45 – DummySimple dictionary (tabular form)

Row	Sequence Number	Format	Name	Child Pointer	Child Count
0	0	set	DummySimple	1	4
1	0	array	ChildArrayProperty	5	1
2	1	string	Id	null	0
3	2	boolean	SampleEnabledProperty	null	0
4	3	integer	SampleIntegerProperty	null	0
5	0	set	null (anonymous array elements)	6	2
6	0	boolean	AnotherBoolean	null	0
7	1	enum	LinkStatus	8	3
8	0	string	LinkDown	null	0
9	1	string	LinkUp	null	0
10	2	string	NoLink	null	0

2078 Finally, in binary form, the dictionary appears as shown in Figure 7. (Colors in this example match those used in
 2079 Figure 5.)

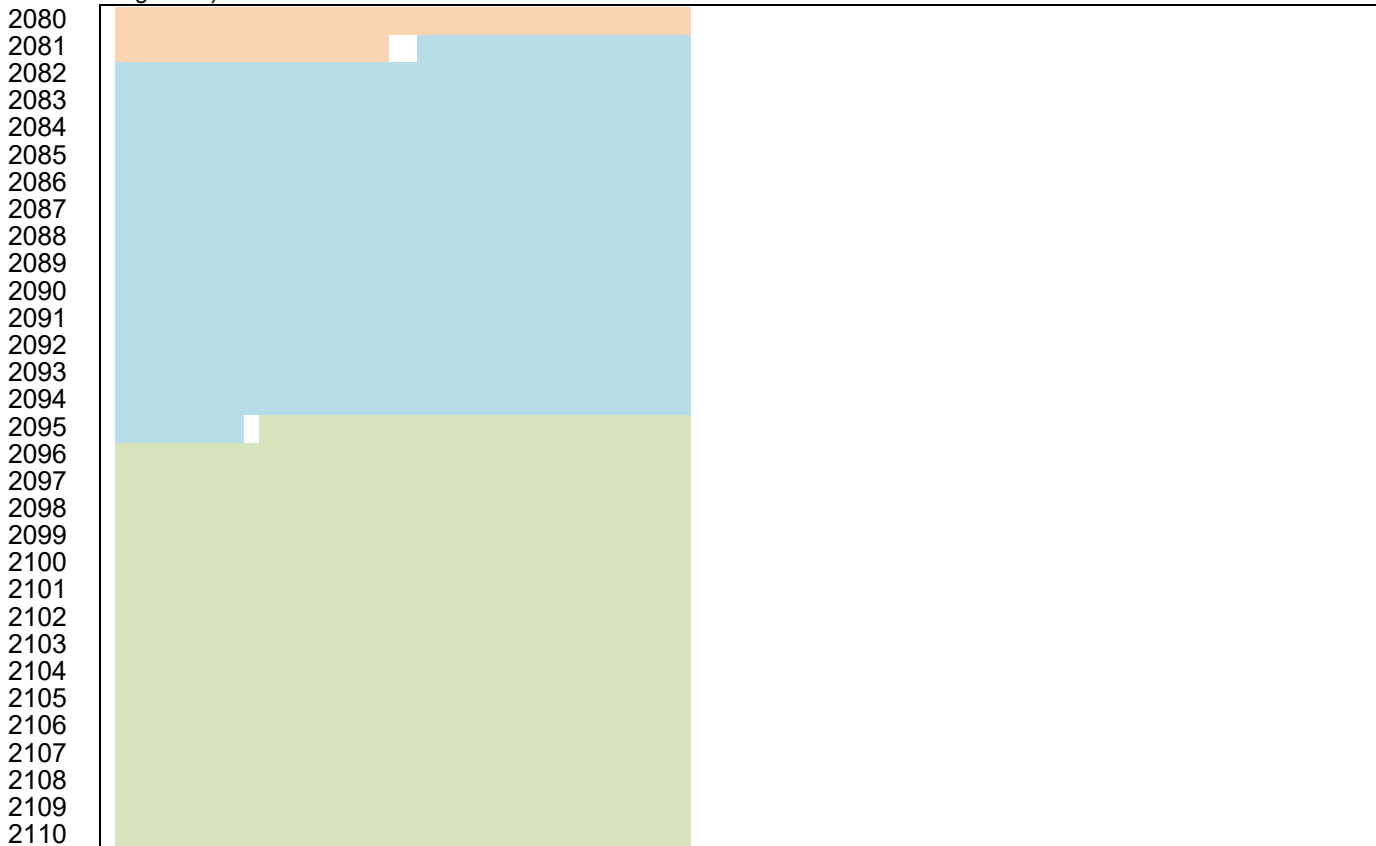


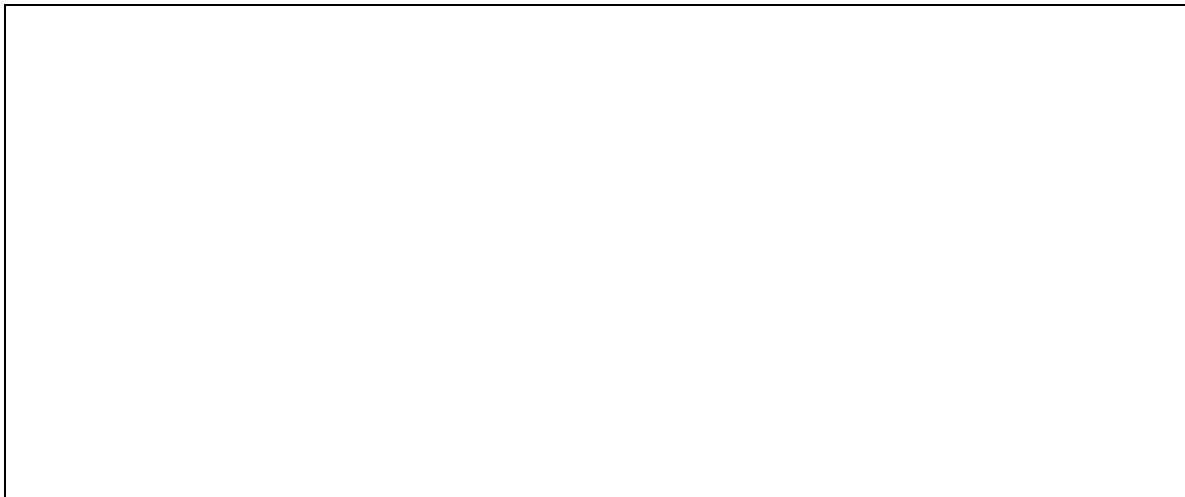


Figure 7 – DummySimple dictionary – binary form

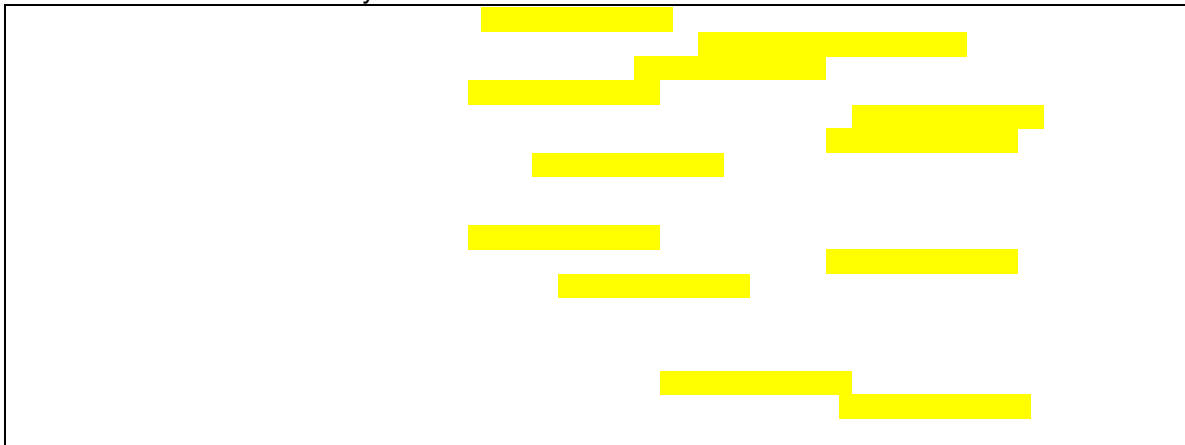
8.6.2 Example encoding

For this example, we start with the following data (shown here in an XML representation).

NOTE The names assigned to array elements are fictitious and inserted for illustrative purposes only. Also, the encoding sequence presented here is only one possible approach; any sequence that generates the same result is acceptable. The value of the @odata.id annotation shown here is a deferred binding (see clause 8.3) that assumes the DummySimple resource corresponds to a Redfish Resource PDR with resource ID 10. Finally, for illustrative purposes we omit here the header bytes contained within the bejEncoding type that are not part of the bejTuple PLDM type.



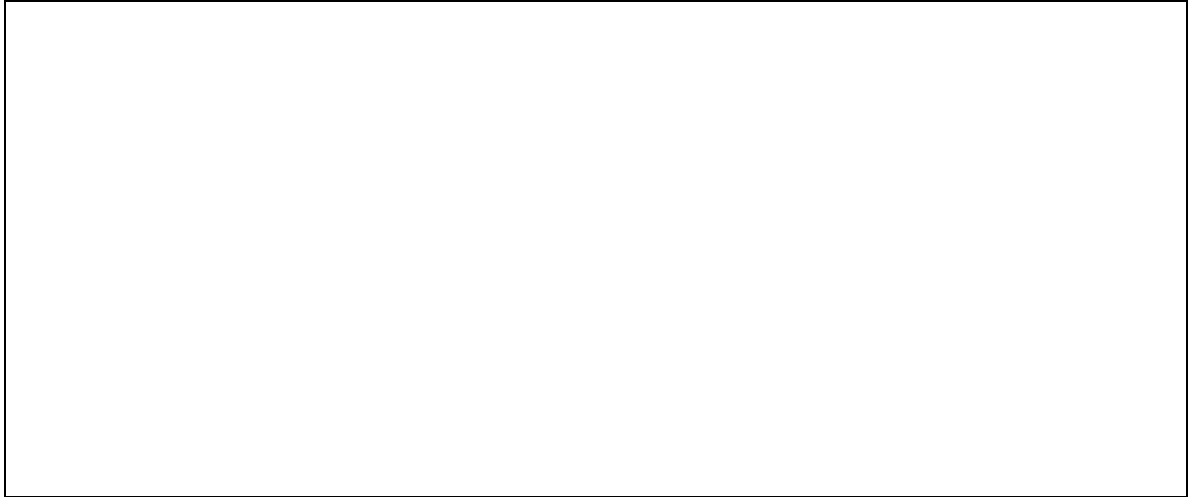
The first step of the encoding process is to insert sequence numbers, which can be retrieved from the relevant dictionary. (For purposes of this example, we are assuming that the @odata.id annotation is sequence number 16 in the annotation dictionary.) Sequence numbers for array elements correspond to their zero-based index within the array.



After the sequence numbers are fully characterized, they can be encoded. We encode which dictionary these sequence numbers came by shifting them left one bit to insert 0b (major dictionary) or 1b (annotation dictionary) as the low order bit per clause 8.2.1. As the sequence numbers are now assigned,

2168 names of properties and enumeration values are no longer needed:

2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187



2188 The next step is to convert everything into BEJ SFLV Tuples. Per clause 5.3.12, the value of an
2189 enumeration is the sequence number for the selected option.

2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204



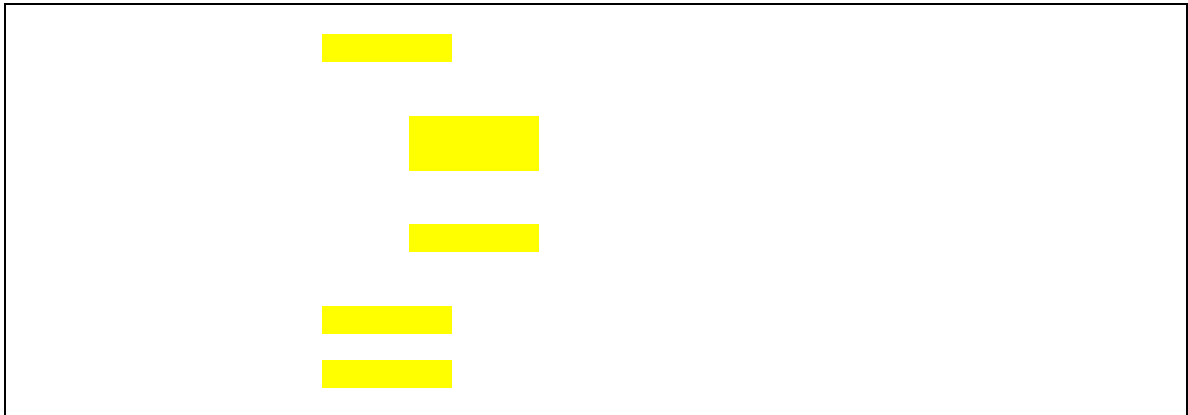
2205 We now encode the formats and the leaf nodes, following Table 9. For sets and arrays, the value
2206 encoding count prefix is a nonnegative Integer; we can encode that now as well per Table 4. Note the null
2207 terminator for the string. The encoded sequence numbers for enumeration values do not need a
2208 dictionary selector inserted as the LSB as the dictionary was already indicated with the sequence number
2209 for the enumeration itself in the format tuple field. The @odata.id annotation string value contains a
2210 deferred binding, so we set that bit in the format tuple field.

2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226



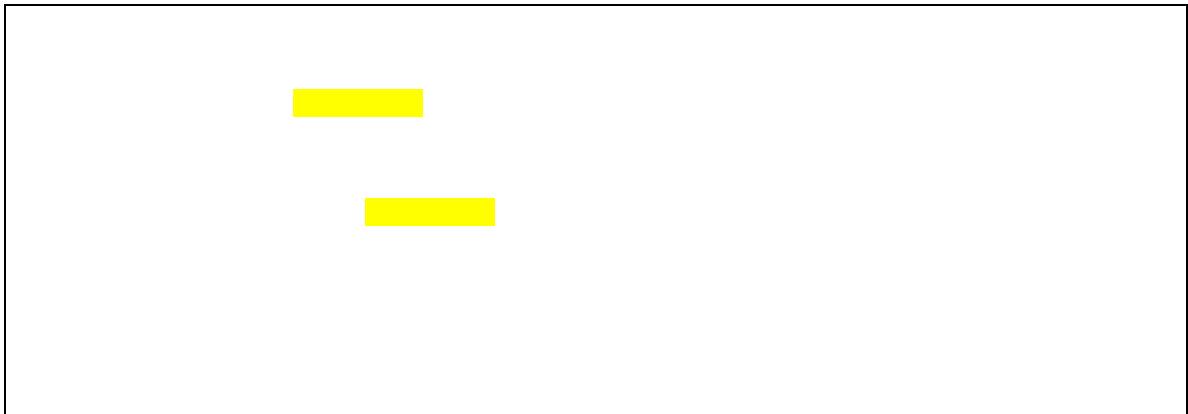
2227 All that remains is to fill in the length values. We begin at the leaves:

2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243



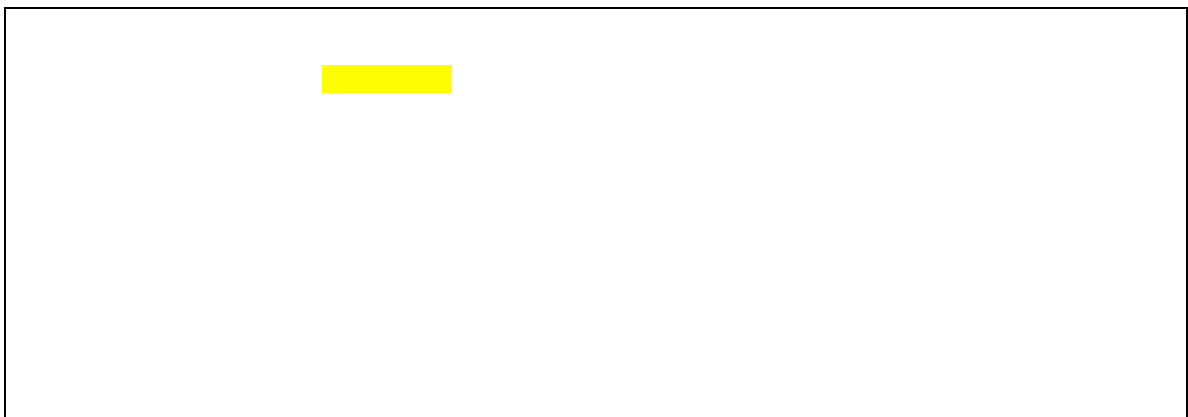
2244 We then work our way from the leaves towards the outermost enclosing tuples. First, the array element sets:

2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261



2262 Next, the array itself:

2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278



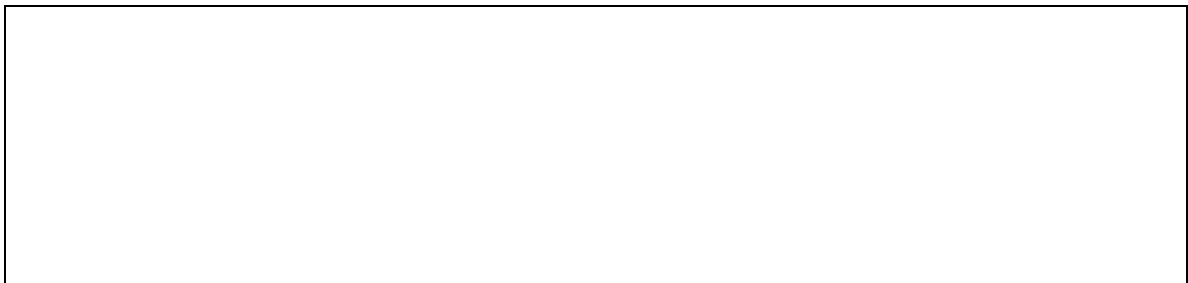
2279 Finally, the outermost set:

2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295



2296 The encoded bytes may now be read off, and the inner encoding is complete:

2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307



2308 **8.6.3 Example decoding**

2309 The decoding process is largely the inverse of the encoding process. For this example, we start with the
2310 final encoded data from clause 8.6.1:

2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321



2322 The first step of the decoding process is to map the byte data to {SFLV} tuples, using the length bytes and
2323 set/array counts to identify tuple boundaries:

2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335



2336
2337
2338
2339



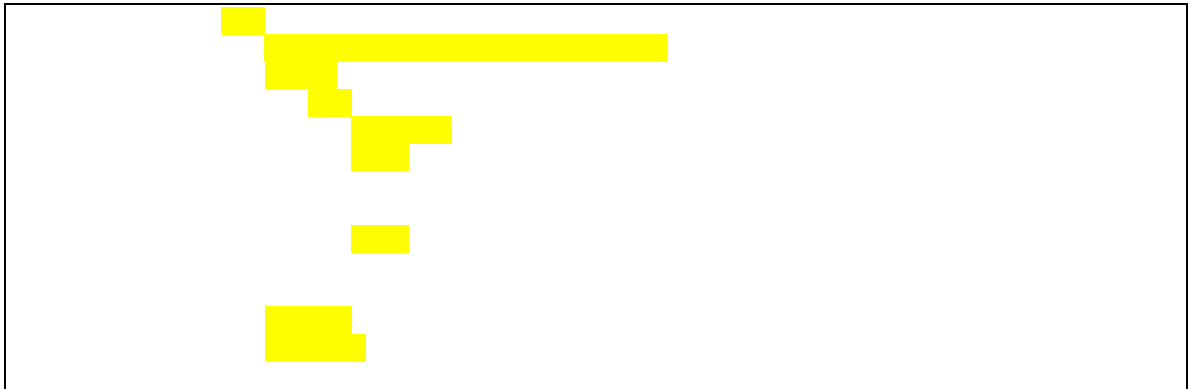
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355

After the tuple boundaries are understood, the length and count data are no longer needed:



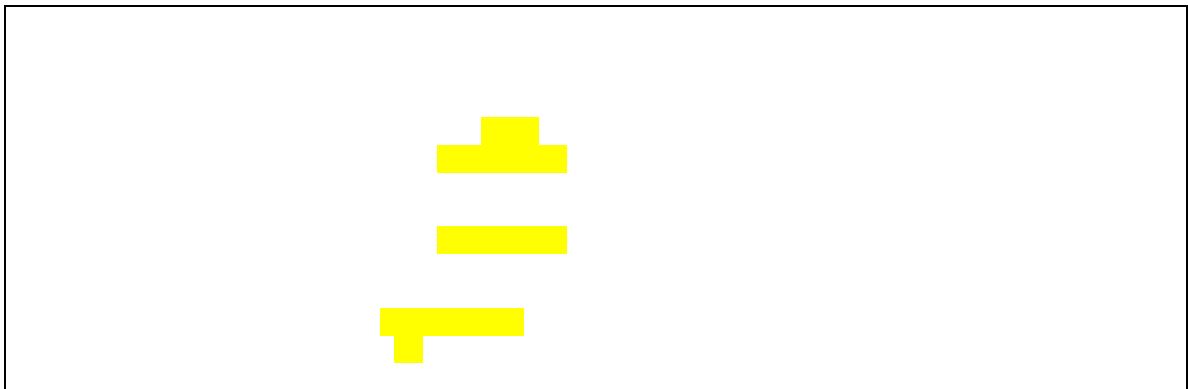
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372

The next step is to decode format tuple bytes using Table 9. This will tell us how to decode the value data:



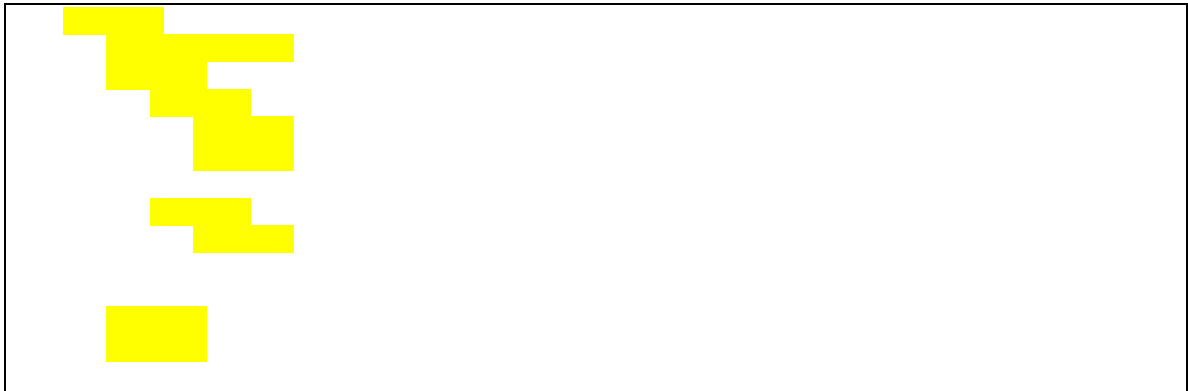
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390

We now decode value data. The deferred binding for the @odata.id property can now be processed, translating from "%L10" to "/redfish/v1/systems/1/DummySimples/1", an instance in a collection of resources of type DummySimple:



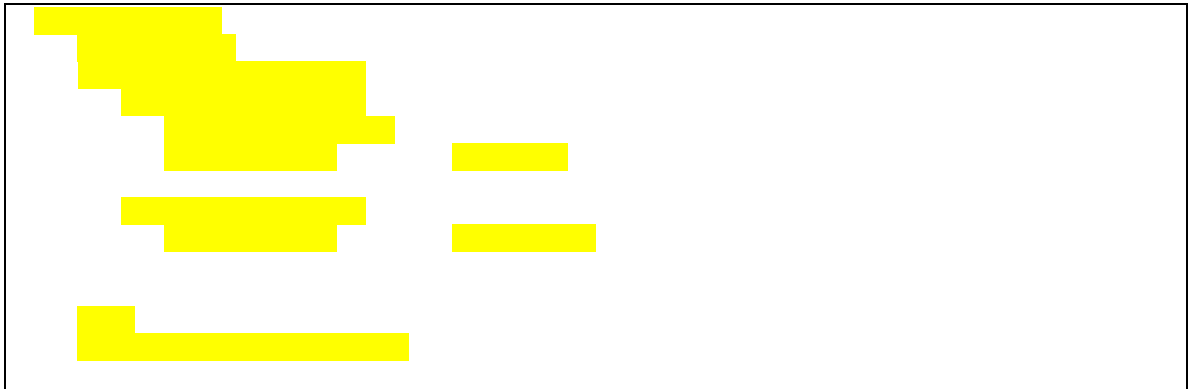
Next, we decode the sequence numbers to identify which dictionary they select:

2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406



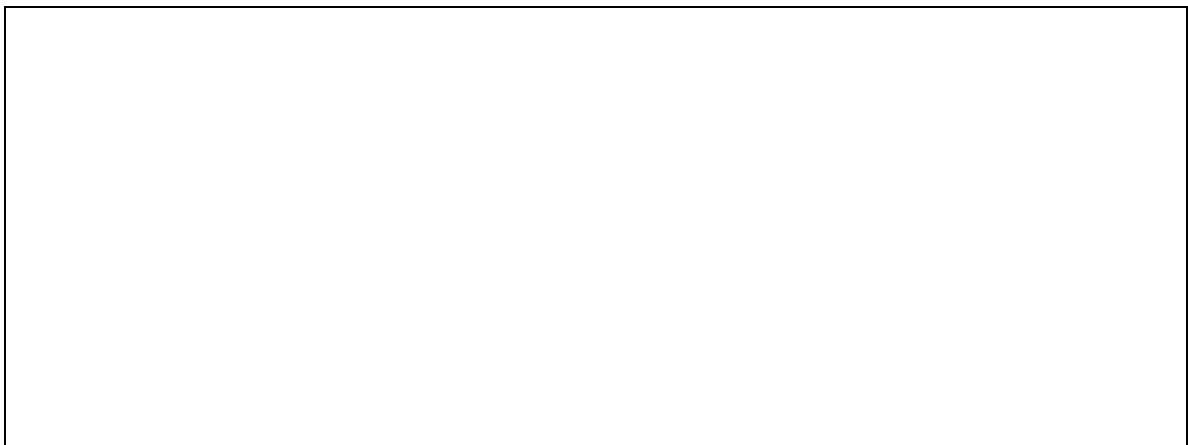
2407 Next, we use the selected dictionary to replace decoded sequence numbers with the strings they
2408 represent:

2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423



2424 We can now write out the decoded BEJ data in JSON format if desired (an MC will need to do this to
2425 forward an RDE Device's response to a client, but an RDE Device may not need this step):

2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442



2443 **8.7 BEJ locators**

2444 A BEJ locator represents a particular location within a resource at which some operation is to take place.
2445 The locator itself consists of a list of sequence numbers for the series of nodes representing the traversal
2446 from the root of the schema tree down to the point of interest. The list of schema nodes is concatenated
2447 together to form the locator. A locator with no sequence numbers targets the root of the schema.

2448 NOTE The sequence numbers are absolute as they are relative to the schema, not to the subset of the schema for
2449 which the RDE Device supports data. This enables a locator to be unambiguous.

2450 As an example, consider a locator, encoded for the example dictionary of clause 8.6.1:

2451 0x01 0x08 0x01 0x00 0x01 0x00 0x01 0x06 0x01 0x02

2452 Decoding this locator, begins with decoding the length in bytes of the locator. In this case, the first two
2453 bytes specify that the remainder of the locator is 8 bytes long. The next step is to decode the bejTupleS-
2454 formatted sequence numbers. The low-order bit of each sequence number references the schema to
2455 which it refers; in this case, the pattern 0b indicates the major schema. Decoding produces the following
2456 list:

2457 0, 0, 3, 1

2458 Now, referring to the dictionary enables identification of the target location. Remember that all indices are
2459 zero-based:

- 2460 • The first zero points to DummySimple
- 2461 • The second zero points to the first child of DummySimple, or ChildArrayProperty
- 2462 • The three points to the fourth element in the ChildArrayProperty array, an anonymous instance
2463 of the array type (array instances are not reflected in the dictionary, but are implicitly the
2464 immediate children of any array)
- 2465 • The one points to the second child inside the ChildArray element type, or LinkStatus

2466 9 Operational behaviors

2467 This clause describes the operational behavior for initialization, Operations/Tasks, and Events.

2468 9.1 Initialization (MC perspective)

2469 The following clauses present initialization of RDE Devices with MCs.

2470 9.1.1 Sample initialization ladder diagram

2471 Figure 8 presents the ladder diagram for an example initialization sequence.

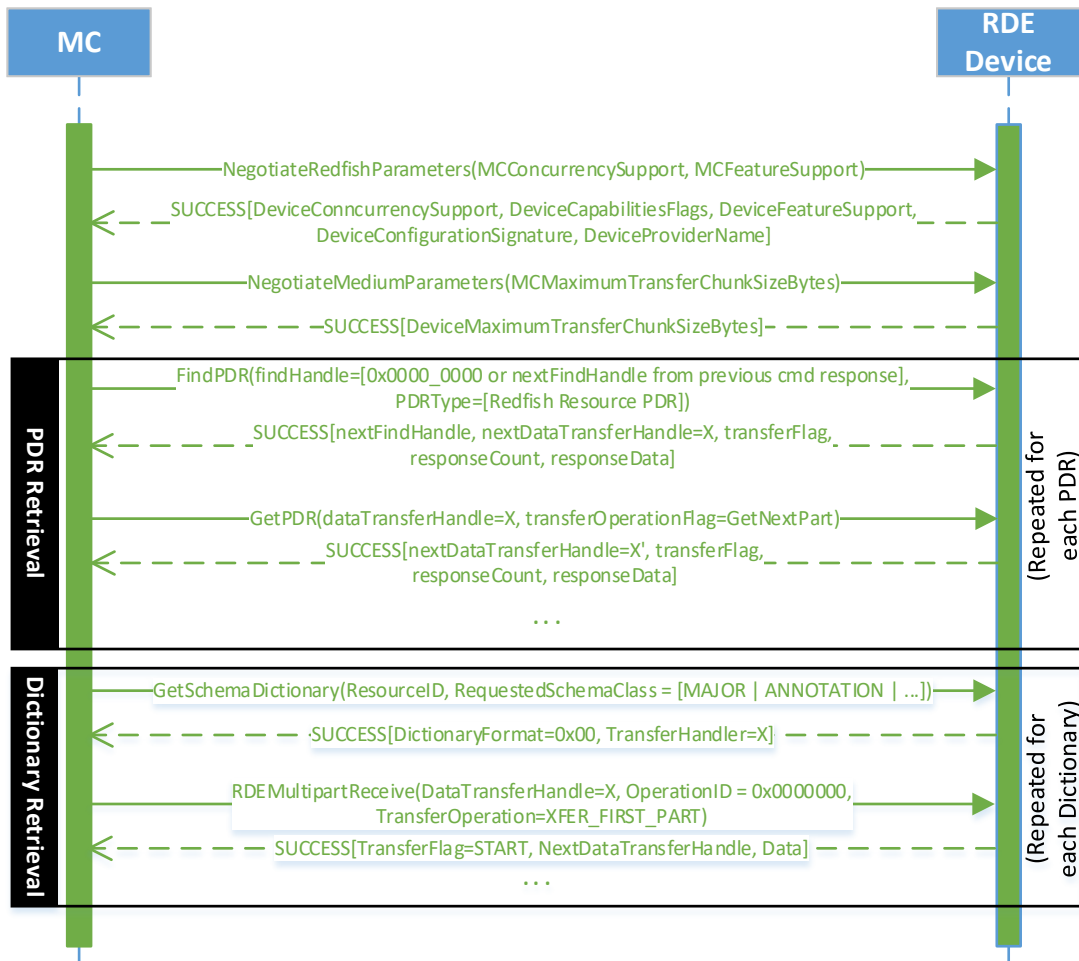
2472 Once the MC detects the RDE Device, it begins the discovery process by invoking the
2473 NegotiateRedfishParameters command to determine the concurrency and feature support for the RDE
2474 Device. It then uses the NegotiateMediumParameters command to determine the maximum message
2475 size that the MC and the RDE Device can both support. This finishes the RDE discovery process.

2476 After discovery comes the RDE registration process. It consists of two parts, PDR retrieval and dictionary
2477 retrieval. To retrieve the RDE PDRs, the MC utilizes the PLDM for Platform Monitoring and Control
2478 FindPDR command to locate PDRs that are specific to RDE⁴. For each such PDR located, the MC then
2479 retrieves it via one or more message sequences in the PLDM for Platform Monitoring and Control
2480 GetPDR command.

2481 After all the PDRs are retrieved, the next step is to retrieve dictionaries. For each Redfish Resource PDR
2482 that the MC retrieved, it retrieves the relevant dictionaries via a standardized process in which it first
2483 executes the GetSchemaDictionary command to obtain a transfer handle for the dictionary. It then uses
2484 the transfer handle with the RDE MultipartReceive command to retrieve the corresponding dictionary.

⁴ Note: FindPDR is an optional command. If the RDE Device does not support it, the MC may achieve equivalent functionality by using GetPDR to transfer of each PDR one at a time, discarding any that are not RDE PDRs.

2485 Multiple initialization variants are possible; for example, it is conceivable that retrieval of some or all
 2486 dictionaries could be postponed until such time as the MC needs to translate BEJ and/or JSON code for
 2487 the relevant schema. Further, the MC may be able to determine that one or more of the dictionaries it has
 2488 already retrieved is adequate to support a PDR and thus skip retrieving that dictionary anew. Finally, if the
 2489 DeviceConfigurationSignature from the NegotiateRedfishParameters command matches the one for data
 2490 that the MC has already cached for the RDE Device, it may skip the retrieval altogether.



2491

2492

Figure 8 – Example Initialization ladder diagram

2493

9.1.2 Initialization workflow diagram

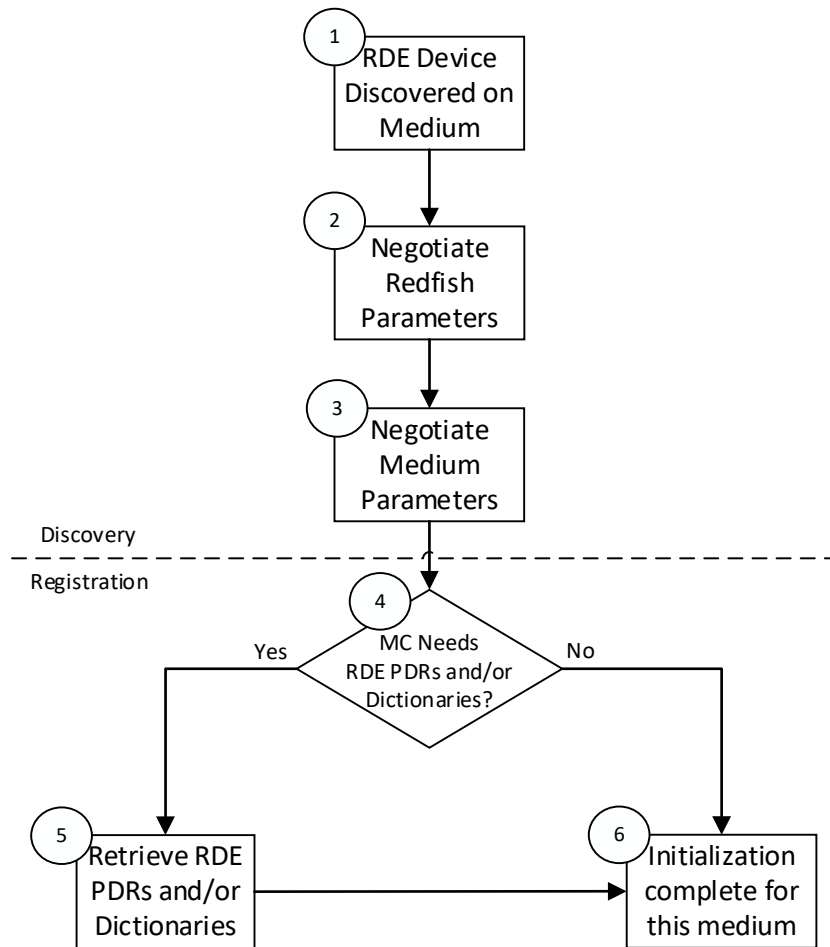
2494

Table 46 details the information presented visually in Figure 9.

2495

Table 46 – Initialization Workflow

Step	Description	Condition	Next Step
1 – DISCOVERY	The MC discovers the presence of the RDE Device through either a medium-specific or other out-of-band mechanism	None	2
2 – NEG_REDFISH	The MC issues the NegotiateRedfishParameters command to the device in order to learn basic information about it	Successful command completion	3
3 – NEG_MEDIUM	The MC issues the NegotiateMediumParameters command to the RDE Device to learn how the RDE Device intends to behave with this medium	Successful command completion	4
4 –NEED_PDR / DICTIONARY_ CHECK	The MC may already have dictionaries and PDRs for the RDE Device cached, such as if this is not the first medium the RDE Device has been discovered on. The MC may choose not to retrieve a fresh copy if the DeviceConfigurationSignature from the NegotiateRedfishParameters command's response message matches what was previously received.	MC does not need to retrieve PDRs or dictionaries for this RDE Device	6
		Otherwise	5
5 – RETRIEVE_PDR / DICTIONARY	The MC retrieves PDRs and/or dictionaries from the RDE Device	Retrieval complete	6
6 – INIT_COMPLETE	The MC has finished discovery and registration for this device	None	None



2496

2497

Figure 9 – Typical RDE Device discovery and registration

2498 **9.2 Operation/Task lifecycle**

2499 The following clauses present the Task lifecycle from two perspectives, first from an Operation-centric
 2500 viewpoint and then from the RDE Device perspective. MC and RDE Device implementations of RDE shall
 2501 comply with the sequences presented here.

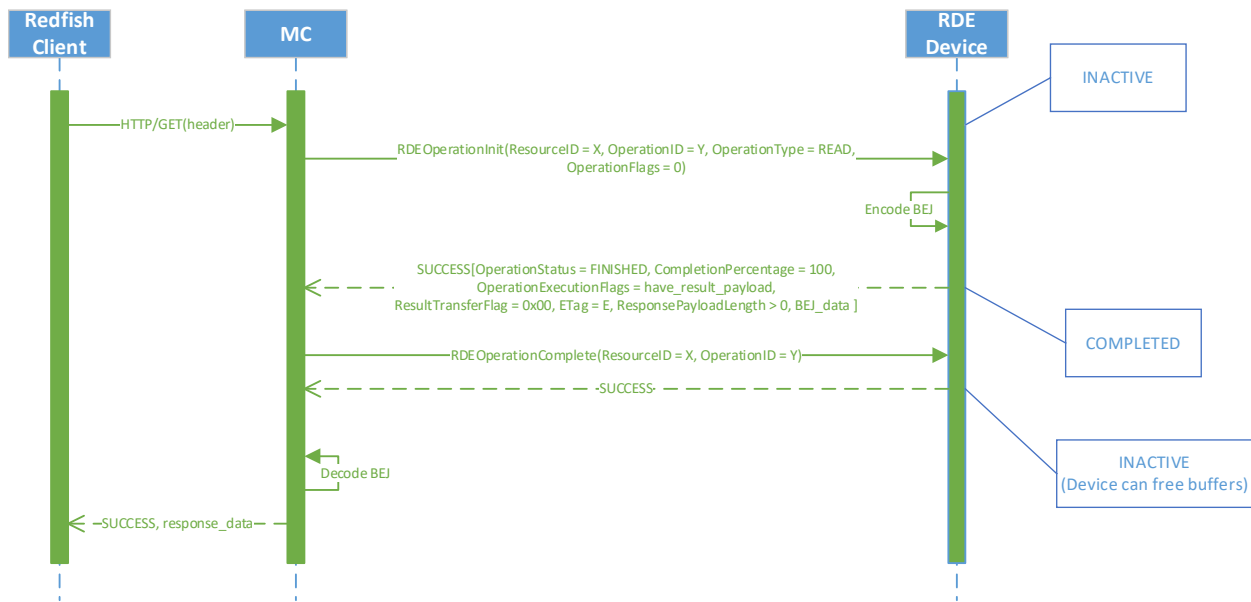
2502 **9.2.1 Example Operation command sequence diagrams**

2503 This clause presents request/response messaging sequences for common Operations.

2504 **9.2.1.1 Simple read Operation ladder diagram**

2505 Figure 10 presents the ladder diagram for a simple read Operation. The Operation begins when the
 2506 Redfish client sends a GET request over an HTTP connection to the MC. The MC decodes the URI
 2507 targeted by the GET operation to pin it down to a specific resource and PDR and sends the
 2508 RDEOperationInit command to the RDE Device that owns the PDR, with OperationType set to READ.
 2509 The RDE Device now has everything it needs for the Operation, so it performs a BEJ encoding of the
 2510 schema data for the requested resource and sends it as an inline payload back to the MC. Sending inline
 2511 is possible in this case because the read data is small enough to not cause the response message to
 2512 exceed the maximum transfer size that was previously negotiated in the NegotiateMediumParameters
 2513 command. The MC in turn has all of the results for the Operation, so it sends RDEOperationComplete to
 2514 finalize the Operation. The RDE Device can now throw away the BEJ encoded read result, so it does so

2515 and responds to the MC with success. Finally, the MC uses the dictionary it previously retrieved from the
 2516 RDE Device to decode the BEJ payload for the read command into JSON data and the MC sends the
 2517 JSON data back to the client.



2518

2519

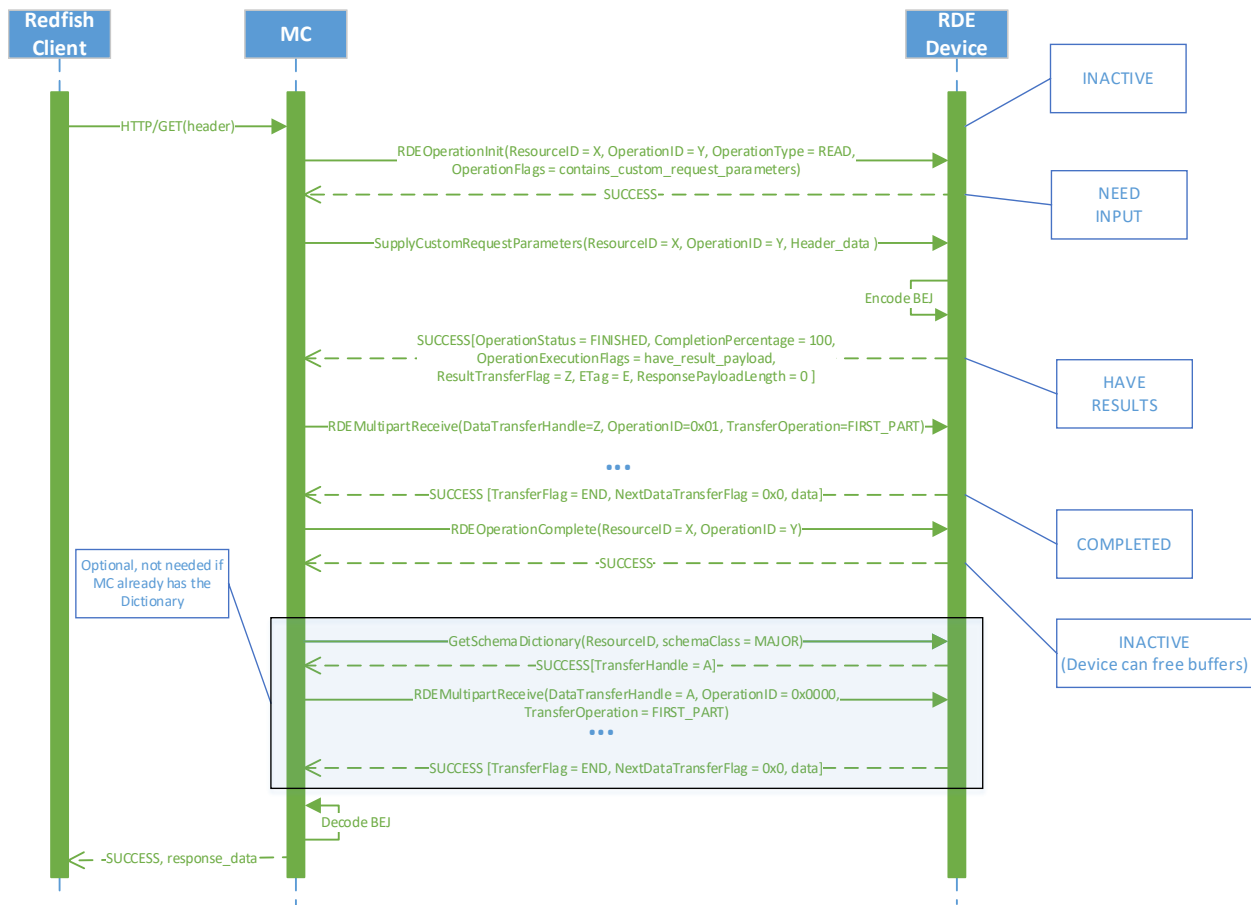
Figure 10 – Simple read Operation ladder diagram

2520 **9.2.1.2 Complex read Operation diagram**

2521 Figure 11 presents the ladder diagram for a more complex read Operation. As with the simple read case,
 2522 the Operation begins when the Redfish client sends a GET request over an HTTP connection to the MC.
 2523 The MC again decodes the URI targeted by the GET operation to pin it down to a specific resource and
 2524 PDR and sends the RDEOperationInit command to the RDE Device that owns the PDR, with
 2525 OperationType set to READ. In this case, however, the OperationFlags that the MC sent with the
 2526 RDEOperationInit command indicate that there are supplemental parameters to be sent to the RDE
 2527 Device, so the RDE Device must wait for these before beginning work on the Operation. The MC sends
 2528 these supplemental parameters to the RDE Device via the SupplyCustomRequestParameters command.

2529 At this point, the RDE Device has everything it needs for the Operation, so just as before, the RDE
 2530 Device performs a BEJ encoding of the schema data for the requested resource. As opposed to the
 2531 previous example, in this case the BEJ-encoded payload is too large to fit within the response message,
 2532 so the RDE Device instead supplied a transfer handle that the MC can use to retrieve the BEJ payload
 2533 separately. The MC, seeing this, performs a series of RDEmultipartReceive commands to retrieve the
 2534 payload. Once it is all transferred, the MC has everything it needs. Whether it needed to retrieve a
 2535 dictionary or it already had one, the MC now sends the RDEOperationComplete command to finalize the
 2536 Operation and allow the RDE Device to throw away the BEJ encoded read result. If the MC needs a
 2537 dictionary to decode the BEJ payload, it may retrieve one via the GetSchemaDictionary command
 2538 followed by one or more RDEmultipartReceive commands to retrieve the binary dictionary data.
 2539 (Normally, the MC would have retrieved the dictionary during initialization; however, if the MC has limited
 2540 storage space to cache dictionaries, it may have been forced to evict it.) Finally, the MC uses the
 2541 dictionary to decode the BEJ payload for the read command into JSON data and then the MC sends the
 2542 JSON data back to the client.
 2543

2544



2545

2546

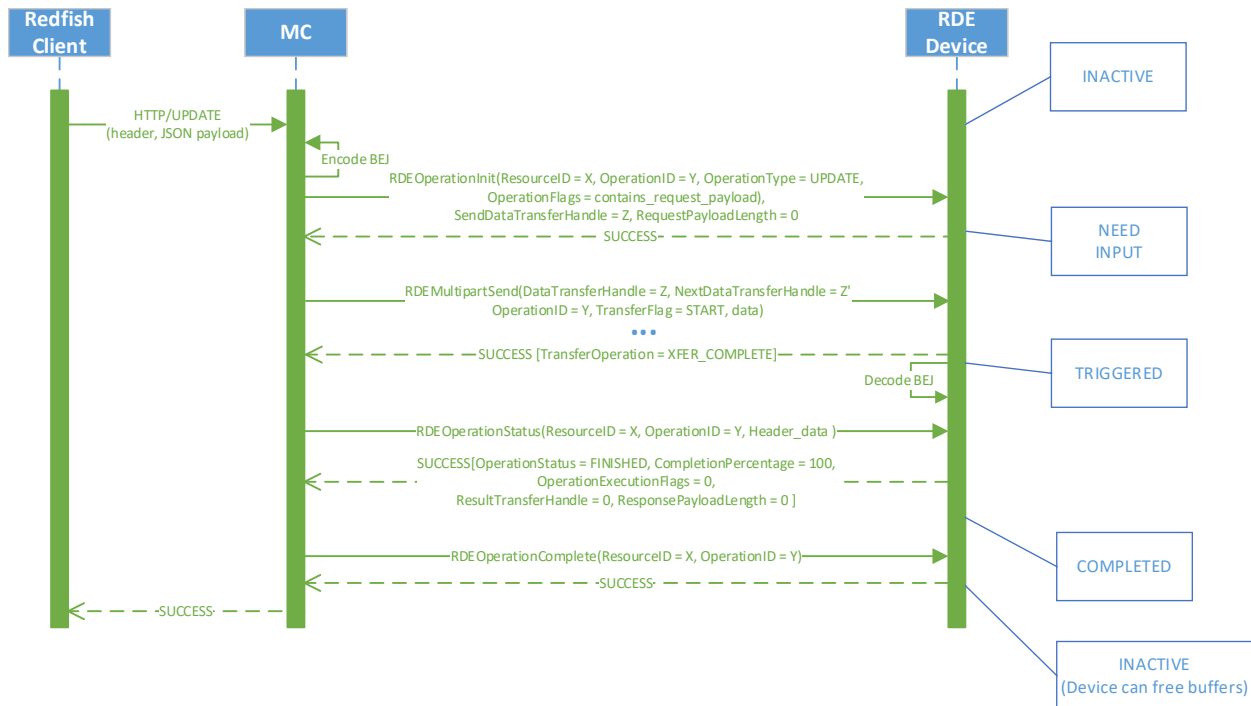
Figure 11 – Complex Read Operation ladder diagram

2547 **9.2.1.3 Write (update) Operation ladder diagram**

2548 Figure 12 presents the ladder diagram for a write Operation. As with the read cases, the Operation begins
 2549 when the Redfish client sends a request over an HTTP connection to the MC, in this case, an UPDATE.
 2550 Once again, the MC decodes the URI targeted by the UPDATE Operation to pin it down to a specific
 2551 resource and PDR. Before it can send the RDEOperationInit command to the RDE Device that owns the
 2552 PDR, the MC must perform a BEJ encoding of the JSON payload it received from the Redfish client. If the
 2553 BEJ encoded payload were small enough to fit within the maximum transfer chunk, the MC could inline it
 2554 with the RDEOperationInit command; however, in this example, that is not the case. The MC therefore
 2555 sends RDEOperationInit with the OperationType set to UPDATE and a nonzero transfer handle. Seeing
 2556 this, the RDE Device knows to expect a larger payload via RDEMultipartSend.

2557 The MC uses the RDEMultipartSend command to transfer the encoded payload to the RDE Device in one
 2558 or more chunks. The contains_request_parameters Operation flag is not set, so the RDE Device will not
 2559 expect supplemental parameters as part of this Operation. Having everything it needs to execute, the
 2560 RDE Device moves to the TRIGGERED state. The MC now sends the RDEOperationStatus command to
 2561 the RDE Device to have it execute the Operation. (In practice, the RDE Device is allowed to begin
 2562 executing the Operation as soon as it has received the request payload, so it may choose not to wait for
 2563 the RDEOperationStatus command to do so.) The RDE Device executes the Operation and sends the

2564 results to the MC as the response to the RDEOperationStatus command. As before, the MC finalizes the
 2565 Operation via RDEOperationComplete and then sends the results back to the client.



2566

2567

Figure 12 – Write Operation ladder diagram

2568 **9.2.1.4 Write (update) with Long-running Task Operation Ladder Diagram**

2569

2570 Figure 13 presents the ladder diagram for a write Operation that spawns a long-running Task. As with the
 2571 previous case, the Operation begins when the Redfish client sends an UPDATE request over an HTTP
 2572 connection to the MC, and the MC decodes the URI targeted by the UPDATE Operation to pin it down to
 2573 a specific resource and PDR. Before it can send the RDEOperationInit command to the RDE Device that
 2574 owns the PDR, the MC must perform a BEJ encoding of the JSON payload it received from the Redfish
 2575 client. Unlike the previous example, the BEJ encoded payload here is small enough to fit in the maximum
 2576 transfer chunk, so the MC inlines it into the RDEOperationInit request command. Again, the
 2577 contains_request_parameters Operation flag is not set, so the RDE Device will not expect supplemental
 2578 parameters as part of this Operation.

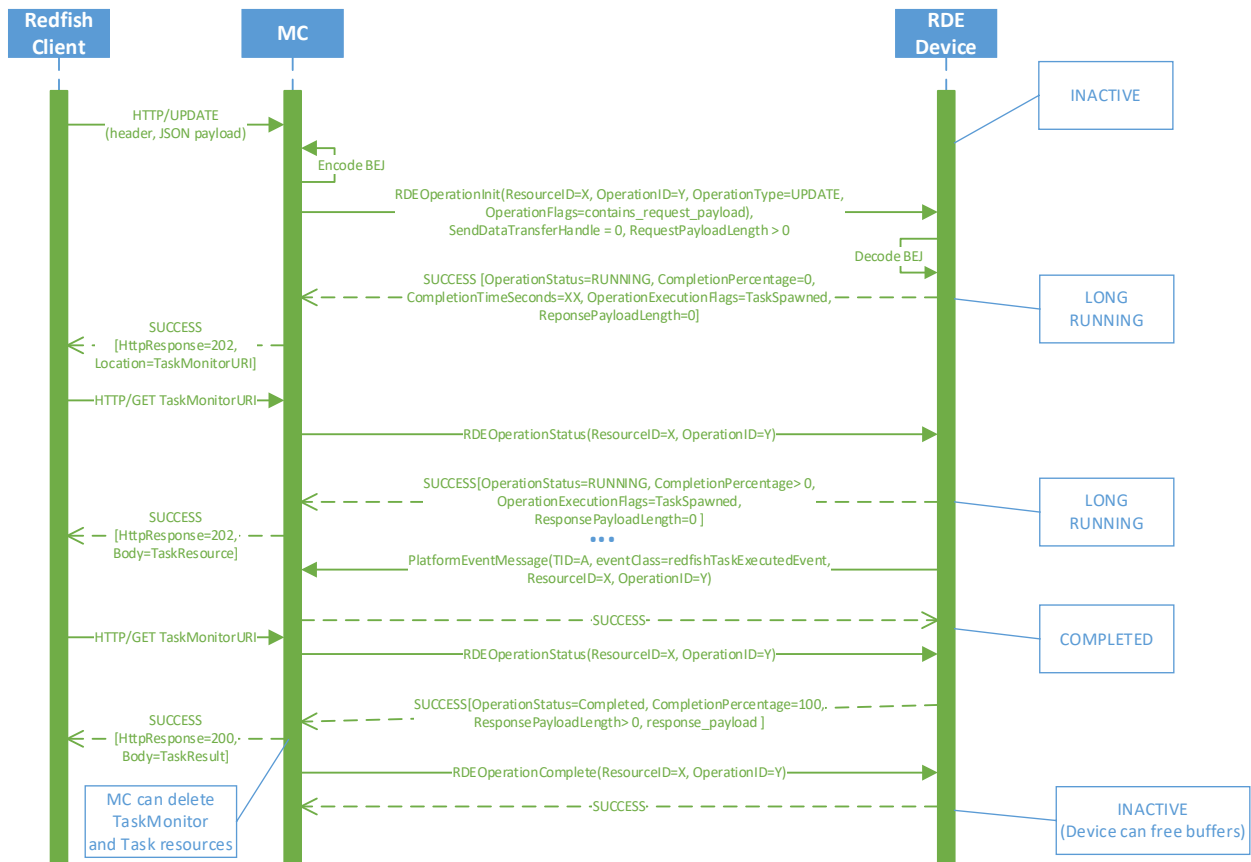
2579 When the RDE Device receives the RDEOperationInit request command, it has everything it needs to
 2580 begin work on the Operation. In this case, the RDE Device determines that performing the write will take
 2581 longer than PT1, so the RDE Device spawns a long-running Task to process the write asynchronously
 2582 and sends TaskSpawned in the OperationExecutionFlags to inform the MC.

2583 When it discovers that the RDE Device spawned a long-running Task, the MC adds a member to the
 2584 Task collection it maintains and synthesizes a TaskMonitor URI to send back to the client in a location
 2585 response header. At this point, the client can issue an HTTP GET to retrieve a status update on the Task;
 2586 when it does so, the MC sends RDEOperationStatus to the RDE Device to get the status update and
 2587 sends it back to the client as the result of the GET operation.

2588 At some point, the asynchronous Task finishes executing. When this happens, the RDE Device issues a
 2589 PlatformEventMessage to send a TaskCompletion event to the MC. (This presupposes that the RDE
 2590 Device and the MC both support asynchronous eventing. Were this not the case, the RDE Device would

2591 still generate the TaskCompletion event, but would wait for the MC to invoke the
 2592 PollForPlatformEventMessage command to report the event.) Regardless of which way the MC gets the
 2593 event, it then sends the RDEOperationStatus command one last time in order to retrieve the final results
 2594 from the Operation. The next time the client performs a GET on the TaskMonitor, the MC can send back
 2595 the final results of the Operation. Finally, the MC finalizes the Operation via RDEOperationComplete at
 2596 which point the MC can delete the Task collection member and the TaskMonitor URI and the RDE Device
 2597 can free up any buffers associated with the Operation and/or Task.

2598



2599

2600 **Figure 13 – Write Operation with long-running Task ladder diagram**

2600

2601 **9.2.2 Operation/Task overview workflow diagrams (Operation perspective)**

2601

2602 This clause describes the operating behavior for MCs and RDE Devices over the lifecycle of Operations
 2603 from an Operation-centric perspective. The workflow diagrams are split between simpler, short-lived
 2604 Operations and those that spawn a Task to be processed asynchronously. These workflow diagrams are
 2605 intended to capture the standard flow for the execution of most Operations, but do not cover every
 2606 possible error condition. For full precision, refer to clause 9.2.3.

2607 **9.2.2.1 Operation overview workflow diagram**

2607

2608 Table 47 details the information presented visually in Figure 14.

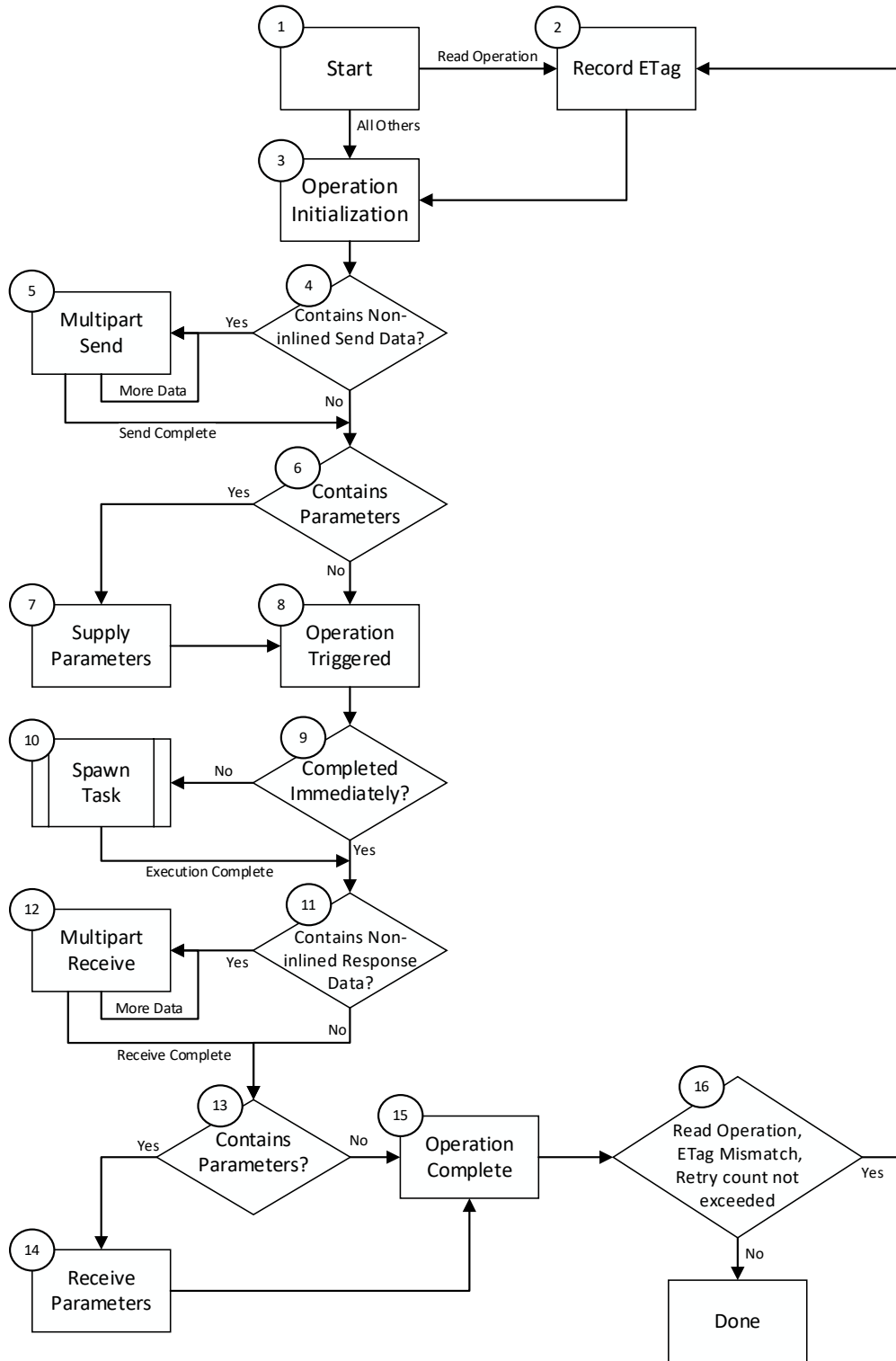
2608

Table 47 – Operation lifecycle overview

Step	Description	Condition	Next Step
1 – START	The lifecycle of an Operation begins when the MC receives an HTTP/HTTPS operation from the client	For any Redfish Read (HTTP/HTTPS GET) operations	2
		For any other operation	3
2 – GET_DIGEST	For Read operations, the MC may use the GetResourceETag command to record a digest snapshot. If the RDE Device advertised that it is capable of reading a resource atomically in the NegotiateRedfishParameters command (see clause 11.1), the MC may skip this step if the read does not span multiple resources (such as through the \$expand request header)	Unconditional	3
3 – INITIALIZE_OP	The MC checks the HTTP/HTTPS operation to see if it contains JSON payload data to be transferred to the RDE Device. If so, it performs a BEJ encoding of this data. It then uses the RDEOperationInit command to begin the Operation with the RDE Device	Unconditional	4
4 – SEND_PAYLOAD_CHK	If the RDE Operation contains BEJ payload data, it needs to be sent to the RDE Device. The payload data may be inlined in the RDEOperationInit request message if the resulting message fits within the negotiated transfer chunk limit.	If the Operation contains a non-inlined payload (that did not fit in the RDEOperationInit request message)	5
		Otherwise	6
5 – SEND_PAYLOAD	The MC uses the RDEmultipartSend command to send BEJ-encoded payload data to the RDE Device	The last chunk of payload data has been sent	6
		More data remains to be sent	5
6 – SEND_PARAMS_CHK	If the RDE Operation contains uncommon request parameters or headers that need to be transferred to the RDE Device, they need to be sent to the RDE Device.	If the Operation contains supplemental request parameters	7
		Otherwise	8
7 – SEND_PARAMS	The MC uses the SupplyCustomRequestParameters command to submit the supplemental request parameters to the RDE Device	Unconditional	8
8 – TRIGGERED	The RDE Device begins executing the Operation as soon as it has all the information it needs for it	Unconditional	9
9 – COMPLETION_CHK	The RDE Device must respond to the triggering command (that provided the last bit of information needed to execute the Operation or a follow-up call to RDEOperationStatus if the last data	If the RDE Device is able to complete the Operation “quickly”	11
		Otherwise	10

Step	Description	Condition	Next Step
	was sent via RDEMultiPartSend) within PT1 time. If it can complete the Operation within that timeframe, it does not need to spawn a Task to run the Operation asynchronously.		
10 – LONG_RUN	If the RDE Device was not able to complete the Operation quickly enough it spawns a Task to execute asynchronously. See Figure 15 for details of the Task sublife cycle.	Once the Task finishes executing	11
11 – RCV_PAYLOAD_CHK	If the Operation contains a response payload, the RDE Device encodes it in BEJ format. If the response payload is small enough to inline and have the response message fit within the negotiated maximum transfer chunk, the RDE Device appends it to the response message of: <ul style="list-style-type: none"> • RDEOperationInit, if this was the triggering command • SupplyCustomRequestParameters, if this was the triggering command • The first RDEOperationStatus after a triggering RDEMultiPartSend command, if the Operation could be completed “quickly” • The first RDEOperationStatus after asynchronous Task execution finishes, otherwise 	If there is no payload or if the payload is small enough to be inlined into the response message of the appropriate command	13
		Otherwise	12
12 – RCV_PAYLOAD	The MC uses the RDEMultiPartReceive command to retrieve the BEJ-encoded payload from the RDE Device	The last chunk of payload data has been sent	13
		More data remains to be sent	12
13 – RCV_PARAMS_CHK	The MC checks to see if the Operation result contains supplemental response parameters	If the Operation contains response parameters	14
		Otherwise	15
14 – RCV_PARAMS	The MC uses the RetrieveCustomResponseParameters command to obtain the supplemental response parameters. NOTE The transfer of a non-inlined response payload and supplemental response parameters may be performed in either order. For simplicity, the flow shown assumes that a response payload would be transferred before	Unconditional	15

Step	Description	Condition	Next Step
	supplemental response parameters; however, the opposite assumption could be made by swapping the positions of blocks 11/12 with blocks 13/14 in the figure.		
15 – COMPLETE	The MC sends the RDEOperationComplete command to finalize the Operation	n/a	n/a
16 – CMP_DIGEST	If the Operation was a read and the MC collected an ETag in step 2, the MC compares the response ETag with the one it collected in step 2 to check for a consistency violation. If it finds one, it may retry the operation or give up. The MC may skip the consistency check (treat it as successful without checking) if the RDE Device advertised that it has the capability to read a resource atomically in its response to the NegotiateRedfishParameters command (see clause 11.1).	Read operation and mismatched ETags and retry count not exceeded	2
		Not a read, no ETag collected, the ETags match, or retry count exceeded	n/a: Done



2610

2611

Figure 14 – RDE Operation lifecycle overview (holistic perspective)

9.2.2.2 Task overview workflow diagram

2613 Table 48 details the information presented visually in Figure 15.

Table 48 – Task lifecycle overview

Current Step	Description	Condition	Next Step
1 – TRIGGERED	The sublifecycle of a Task begins when the RDE Device receives all the data it needs to perform an Operation. (This corresponds to Step 8 in Table 47.)	Unconditional	2
2 – COMPLETION_CHK	The RDE Device must respond to the triggering command (that provided the last bit of information needed to execute the Operation) within PT1 time. If it cannot complete the Operation within that timeframe, it spawns a Task to run the Operation asynchronously.	If the RDE Device is able to complete the Operation quickly (not a Task)	17
		Otherwise	3
3 – LONG_RUN	The RDE Device runs the Task asynchronously	Unconditional	5
4 – REQ_STATUS	The MC may issue an RDEOperationStatus command at any time to the RDE Device.	If issued	5
5 –STATUS_CHK	The RDE Device must be ready to respond to an RDEOperationStatus command while running a Task asynchronously	Status request received	6
		No status request received	8
6 – PROCESS_STATU S	The RDE Device sends a response to the RDEOperationStatus command to provide a status update	Unconditional	3
7 – REQ_KILL	The MC may issue an RDEOperationKill command at any time to the RDE Device	Unconditional	8
8 –KILL_CHK	The RDE Device must be ready to respond to an RDEOperationKill command while running a Task asynchronously	Kill request received	9
		No kill request received	10
9 – PROCESS_KILL	If the RDE Device receives a kill request, it may or may not be able to abort the Task. This is an RDE Device-specific decision about whether the Task has crossed a critical boundary and must be completed	RDE Device cannot stop the Task	10
		RDE Device can stop the Task	11
10 – ASYNC_EXECUTE_ FINISHED_CHK	The RDE Device should eventually complete the Task	If the Task has been completed	12
		If the Task has not been completed	3
11 – PERFORM_ABORT	The RDE Device aborts the Task in response to a request from the MC	Unconditional	17

Current Step	Description	Condition	Next Step
12 – COMPLETION_EVENT	After the Task is complete, the RDE Device generates a Task Completion Event	Unconditional	13
13 – ASYNC_CHK	The mechanism by which the Task completion Event reaches the MC depends on how the MC configured the RDE Device for Events via the PLDM for Platform Monitoring and Control SetEventReceiver command	Asynchronous Events	14
		Polled Events	15
14 – PEM_POLL	The MC uses the PollForPlatformEventMessage command to check for Events and finds the Task Completion Event	Unconditional	16
15 – PEM_SEND	The RDE Devices sends the Task Completion Event to the MC asynchronously via the PlatformEventMessage command	Unconditional	16
16 – GET_TASK_FOLLOWUP	After receiving the Task completion Event, the MC uses the RDEOperationStatus command to retrieve the outcome of the Task's execution	Unconditional	17
17 – TASK_DONE	The MC checks the response message to the RDEOperationStatus command to see if there is a response payload (This corresponds to Step 11 in Table 47.)	See Step 11 in Table 49	See Step 11 in Table 49

2615

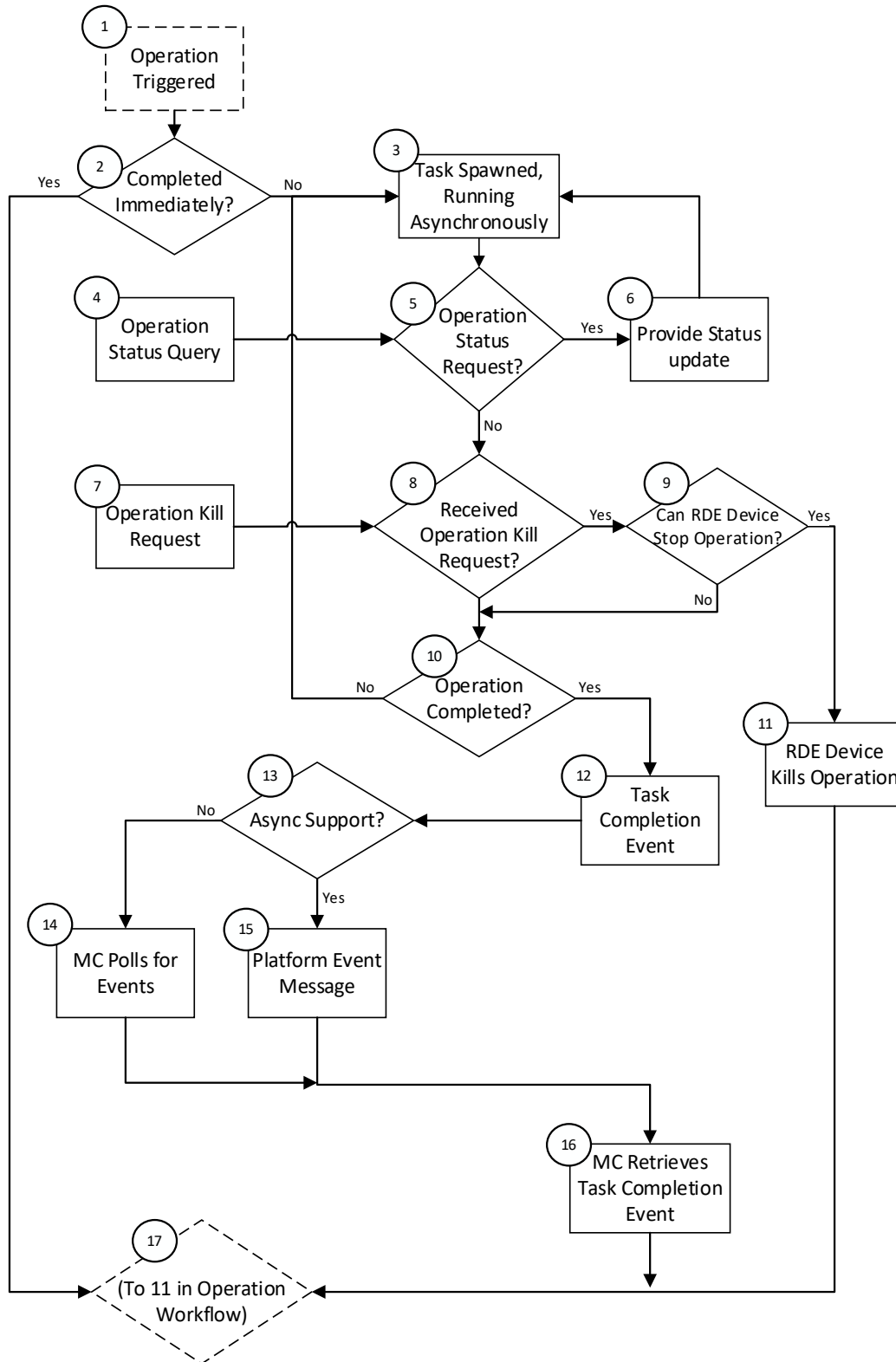


Figure 15 – RDE Task lifecycle overview (holistic perspective)

2616

2617

2618

2619 9.2.3 RDE Operation state machine (RDE Device perspective)

2620 The following clauses describe the operating behavior for the lifecycle of Operations and Tasks from an
2621 RDE Device-centric perspective. Table 49 details the information presented visually in Figure 16. The
2622 states presented in this state machine are not (collectively) the total state for the RDE Device, but rather
2623 the state for the Operation. The total state for the RDE Device would involve separate instances of the
2624 Task/Operation state machine replicated once for each of the concurrent Operations that the RDE Device
2625 and the MC negotiated to support at registration time.

2626 9.2.3.1 State definitions

2627 The following states shall be implemented by the RDE Device for each Operation it is supporting:

- 2628 • INACTIVE
 - 2629 – INACTIVE is the default Operation state in which the RDE Device shall start after
2630 initialization. In this state, the RDE Device is not processing an Operation as it has not
2631 received an RDEOperationInit command from the MC.
- 2632 • NEED_INPUT
 - 2633 – After receiving the RDEOperationInit command, the RDE Device moves to this state if it is
2634 expecting additional Operation-specific parameters or a payload that was not inlined in the
2635 RDEOperationInit command.
- 2636 • TRIGGERED
 - 2637 – Once the RDE Device receives everything it needs to execute an Operation, it begins
2638 executing it immediately. If the triggering command – the command that supplied the last
2639 bit of data needed to execute the Operation – was RDEOperationInit or
2640 SupplyCustomRequestParameters, the response message to the triggering command
2641 reflects the initial results for the Operation. However, if the triggering command was a
2642 RDEMultipartSend, initial results are deferred until the MC invokes the
2643 RDEOperationStatus command. This state captures the case where the Operation was
2644 triggered by a RDEMultipartSend and the MC has not yet sent an RDEOperationStatus
2645 command to get initial results. In this state, the RDE Device may execute the Operation;
2646 alternatively, it may wait to receive RDEOperationStatus to begin execution.
- 2647 • TASK_RUNNING
 - 2648 – If the RDE Device cannot complete the Operation within the timeframe needed for the
2649 response to the command that triggered it, the RDE Device spawns a Task in which to
2650 execute the Operation asynchronously.
- 2651 • HAVE_RESULTS
 - 2652 – When execution of the Operation produces a response parameters or a response payload
2653 that does not fit in the response message for the command that triggered the Operation (or
2654 detected its completion, if a Task was spawned or if there was a payload but no custom
2655 request parameters), the RDE Device remains in this state until the MC has collected all of
2656 these results.
- 2657 • COMPLETED
 - 2658 – The RDE Device has completed processing of the Operation and awaits acknowledgment
2659 from the MC that it has received all Operation response data. This acknowledgment is
2660 done by the MC issuing the RDEOperationComplete command. When the RDE Device
2661 receives this command, it may discard any internal records or state it has maintained for
2662 the Operation.
- 2663 • FAILED

- 2664 – The MC has explicitly killed the Operation or an error prevented execution of the
- 2665 Operation.
- 2666 • ABANDONED
- 2667 – If MC fails to progress the Operation through this state machine, the RDE Device may
- 2668 abort the Operation and mark it as abandoned.

2669 **9.2.3.2 Operation lifecycle state machine**

2670 Figure 16 illustrates the state transitions the RDE Device shall implement. Each bubble represents a
 2671 particular state as defined in the previous clause. Upon initialization, system reboot, or an RDE Device
 2672 reset the RDE Device shall enter the INACTIVE state.

2673 **Table 49 – Task lifecycle state machine**

Current State	Trigger	Response	Next State
0 - INACTIVE	RDEOperationInit - RDE Device not ready - RDE Device does not wish to specify a deferral timeframe	ERROR_NOT_READY, HaveCustomResponseParameters bit in OperationExecutionFlags not set	INACTIVE
	RDEOperationInit - RDE Device not ready - RDE Device does wish to specify a deferral timeframe	ERROR_NOT_READY, HaveCustomResponseParameters bit in OperationExecutionFlags set	HAVE_RESULTS
	RDEOperationInit, SupplyCustomRequestParameters, RDEOperationStatus, RDEOperationKill, or RDEOperationComplete - Resource ID does not correspond to any active Operation	ERROR_NO_SUCH_RESOURCE	INACTIVE
	RDEOperationInit, wrong resource type for POST Operation in request (e.g., Action sent to a collection)	ERROR_WRONG_LOCATION_TYPE	INACTIVE
	RDEOperationInit, RDE Device does not allow the requested Operation	ERROR_NOT_ALLOWED	INACTIVE
	RDEOperationInit, RDE Device does not support the requested Operation	ERROR_UNSUPPORTED	INACTIVE
	RDEOperationInit, Operation ID has MSBit clear (indicating that the MC is attempting to initiate an Operation with an ID reserved for the RDE Device)	ERROR_INVALID_DATA	INACTIVE
	RDEOperationInit, request contains any other error	Various, depending on the specific error encountered	INACTIVE
	RDEOperationStatus	OPERATION_INACTIVE	INACTIVE

Current State	Trigger	Response	Next State
	RDEOperationInit; - valid request - Operation Flags indicate request non-inlined payload or parameters to be sent from MC to RDE Device	Success	NEED_INPUT
	RDEOperationInit; - valid request - Operation Flags indicate no request payload to be sent from MC to RDE Device (or request payload inlined in RDEOperationInit request message) - request flags indicate no supplemental parameters needed - RDE Device cannot complete Operation within PT1	Success	TASK_RUNNING
	RDEOperationInit; - valid request - Operation Flags indicate no request payload to be sent from MC to RDE Device (or request payload inlined in RDEOperationInit request message) - request flags indicate no supplemental parameters needed - RDE Device completes Operation within PT1 - response flags indicate response parameters or a non-inlined response payload to be retrieved from RDE Device	Success	HAVE_RESULTS
	RDEOperationInit; - valid request - Operation Flags indicate no request payload to be sent from MC to RDE Device (or request payload	ERROR_INSUFFICIENT_STORAGE	FAILED

Current State	Trigger	Response	Next State
	inlined in RDEOperationInit request message) <ul style="list-style-type: none"> - request flags indicate no supplemental parameters needed - RDE Device completes Operation within PT1 - response flags indicate response parameters or a non-inlined response payload to be retrieved from RDE Device - Response payload too large for RDE Device to process 		
	RDEOperationInit; <ul style="list-style-type: none"> - valid request - Operation Flags indicate no request payload to be sent from MC to RDE Device (or request payload inlined in RDEOperationInit request message) - request flags indicate no supplemental parameters needed - RDE Device completes Operation within PT1 - no payload to be retrieved from RDE Device or response payload fits within response message such that total response message size is within negotiated maximum transfer chunk - no response parameters 	Success	COMPLETED
	RDEOperationKill (any combination of flags)	ERROR_UNEXPECTED	INACTIVE
	Any other Operation command	ERROR	INACTIVE
1- NEED_INPUT	RDEOperationInit, same rdeOpID	ERROR_OPERATION_EXISTS; no disruption to existing Operation	NEED_INPUT
	RDEOperationInit, different rdeOpID	Success or ERROR_CANNOT_CREATE_OPERATION, depending on whether the RDE Device has another slot to execute an Operation	The new Operation is tracked in a separate copy of the state machine; this Operation remains in NEED_INPUT
	RDEOperationInit request flags indicated supplemental parameters and or payload data to be sent;	None	ABANDONED

Current State	Trigger	Response	Next State
	T _{abandon} timeout waiting for RDEMultipartSend/SupplyCustomRequestParameterscommand		
	RDEOperationKill; - neither run_to_completion nor discard_record flag set	Success	FAILED
	RDEOperationKill; - run_to_completion flag not set - discard_record flag set	Success	INACTIVE
	RDEOperationKill; - run_to_completion flag set - discard_record flag not set	ERROR_INVALID_DATA	NEED_INPUT
	RDEOperationKill; - both run_to_completion and discard_record flags both set	ERROR_UNEXPECTED (can't run to completion without further input from MC, so the request is contradictory)	NEED_INPUT
	RDEOperationStatus	OPERATION_NEED_INPUT	NEED_INPUT
	RDEMultipartSend; - data inlined or Operation flags indicate no payload data	ERROR_UNEXPECTED	NEED_INPUT
	RDEMultipartSend; - transfer error	Error specific to type of transfer failure encountered	NEED_INPUT (MC may retry send or use RDEOperationKill to abort Operation)
	RDEMultipartSend; - more data to be sent from the MC to the RDE Device after this chunk	Success	NEED_INPUT
	RDEMultipartSend; - no more data to be sent from the MC to the RDE Device after this chunk - RDEOperationInit request flags indicated supplemental parameters needed - params not yet sent	Success	NEED_INPUT
	RDEMultipartSend; - no more data to be sent after this chunk - RDEOperationInit request flags indicated supplemental parameters	ERROR_INSUFFICIENT_STORAGE	FAILED

Current State	Trigger	Response	Next State
	not needed or parameters already sent - request payload too large to be processed by RDE Device		
	RDEMultipartSend; - no more data to be sent after this chunk - RDEOperationInit request flags indicated supplemental parameters not needed or parameters already sent	Success	TRIGGERED
	RDEMultipartSend; - data already transferred	ERROR_UNEXPECTED	NEED_INPUT
	SupplyCustomRequestParameters; - Operation includes unsupported ETag operation or query option	ERROR_UNSUPPORTED	FAILED
	SupplyCustomRequestParameters; - Operation flags indicated supplemental parameters not needed or payload data remaining to be sent	ERROR_UNEXPECTED	NEED_INPUT
	SupplyCustomRequestParameters; - no payload data remaining to be sent - ETagOperation is ETAG_IF_MATCH and no ETag matches or ETagOperation is ETAG_IF_NONE_MATCH and an ETag matches	ERROR_ETAG_MATCH	FAILED
	SupplyCustomRequestParameters; - request contains unsupported RDE custom header	ERROR_UNRECOGNIZED_CUSTOM_HEADER	FAILED
	SupplyCustomRequestParameters; - no payload data remaining to be sent - Error occurs in processing of Operation	Error specific to type of failure encountered	FAILED
	SupplyCustomRequestParameters; - no payload data remaining to be sent - RDE Device cannot complete Operation within PT1	Success	LONG_RUNNING

Current State	Trigger	Response	Next State
	SupplyCustomRequestParameters; <ul style="list-style-type: none"> - no payload data remaining to be sent - RDE Device completes Operation within PT1 - response flags indicate response parameters or a non-inlined response payload to be retrieved from RDE Device - response payload too large to be processed by RDE Device 	ERROR_INSUFFICIENT_STORAGE	FAILED
	SupplyCustomRequestParameters; <ul style="list-style-type: none"> - no payload data remaining to be sent - RDE Device completes Operation within PT1 - response flags indicate response parameters or a non-inlined response payload to be retrieved from RDE Device 	Success	HAVE_RESULTS
	SupplyCustomRequestParameters; <ul style="list-style-type: none"> - no payload data remaining to be sent - RDE Device completes Operation within PT1 - no payload to be retrieved from RDE Device or response payload fits within response message such that total response message size is within negotiated maximum transfer chunk - no response parameters 	Success	COMPLETED
	RDEMultipartReceive, RDEOperationComplete	ERROR_UNEXPECTED	NEED_INPUT
	Any other Operation command	ERROR	NEED_INPUT
	2 - TRIGGERED	RDEOperationInit, same rdeOpID	ERROR_OPERATION_EXISTS; no disruption to existing Operation
	RDEOperationInit, different rdeOpID	Success or ERROR_CANNOT_CREATE_OPERATION, depending on whether the RDE Device has another slot to execute an Operation	The new Operation is tracked in a separate copy of the state machine; this Operation remains in TRIGGERED

Current State	Trigger	Response	Next State
	Tabandon timeout waiting for RDEOperationStatus command	None	ABANDONED
	RDEOperationStatus; error occurs in processing of Operation	Error specific to type of failure encountered	FAILED
	RDEOperationKill <ul style="list-style-type: none"> - discard_results flag set - any other flag set 	ERROR_INVALID_DATA	TRIGGERED
	RDEOperationKill <ul style="list-style-type: none"> - discard_results flag set - no other flag set 	ERROR_UNEXPECTED	TRIGGERED
	RDEOperationKill <ul style="list-style-type: none"> - run_to_completion flag set - discard_record flag not set 	ERROR_INVALID_DATA	TRIGGERED
	RDEOperationKill; <ul style="list-style-type: none"> - Operation executing; Operation can be killed - neither run_to_completion nor discard_record flag set 	Success	FAILED
	RDEOperationKill <ul style="list-style-type: none"> - Operation executing - Operation can be killed - run_to_completion flag not set - discard_record flag set 	Success	INACTIVE
	RDEOperationKill <ul style="list-style-type: none"> - Operation executing - Operation can be killed - both run_to_completion and discard_record flags set 	ERROR_UNEXPECTED (can't run to completion without further input from MC to move it to TASK_RUNNING, so the request is contradictory)	TRIGGERED
	RDEOperationKill <ul style="list-style-type: none"> - Operation executing - Operation cannot be killed or Operation execution finished - any combination of run_to_completion and discard_record flags set 	ERROR_OPERATION_UNKILLABLE	TRIGGERED
	RDEOperationStatus; <ul style="list-style-type: none"> - RDE Device cannot complete Operation within PT1 	OPERATION_TASK_RUNNING	TASK_RUNNING
	RDEOperationStatus; <ul style="list-style-type: none"> - RDE Device completes Operation within PT1 - response payload too large for RDE Device to process 	ERROR_INSUFFICIENT_STORAGE	FAILED

Current State	Trigger	Response	Next State
	RDEOperationStatus; <ul style="list-style-type: none"> - RDE Device completes Operation within PT1 - payload to be retrieved from RDE Device or response parameters present 	Success	HAVE_RESULTS
	RDEOperationStatus; <ul style="list-style-type: none"> - RDE Device completes Operation within PT1 - no payload or payload fits within response message such that total response message size is within negotiated maximum transfer chunk - no response parameters 	Success	COMPLETED
	RDEMultipartSend, RDEMultipartReceive, SupplyCustomRequestParameters, RetrieveCustomResponseParameters, RDEOperationComplete	ERROR_UNEXPECTED	TRIGGERED
	Any other Operation command	ERROR	TRIGGERED
3 - TASK_RUNNING	RDEOperationInit, same rdeOpID	ERROR_OPERATION_EXISTS; no disruption to existing Operation	TASK_RUNNING
	RDEOperationInit, different rdeOpID	Success or ERROR_CANNOT_CREATE_OPERATION, depending on whether the RDE Device has another slot to execute an Operation	The new Operation is tracked in a separate copy of the state machine; this Operation remains in TASK_RUNNING
	Error occurs in processing of Operation	None	FAILED
	RDEOperationKill <ul style="list-style-type: none"> - discard_results flag set - any other flag set 	ERROR_INVALID_DATA	TASK_RUNNING
	RDEOperationKill <ul style="list-style-type: none"> - discard_results flag set - no other flag set 	ERROR_UNEXPECTED	TASK_RUNNING
	RDEOperationKill <ul style="list-style-type: none"> - run_to_completion flag set - discard_record flag not set 	ERROR_INVALID_DATA	TASK_RUNNING
	RDEOperationKill; <ul style="list-style-type: none"> - Operation can be aborted - neither run_to_completion nor discard_record flag set 	Success	FAILED

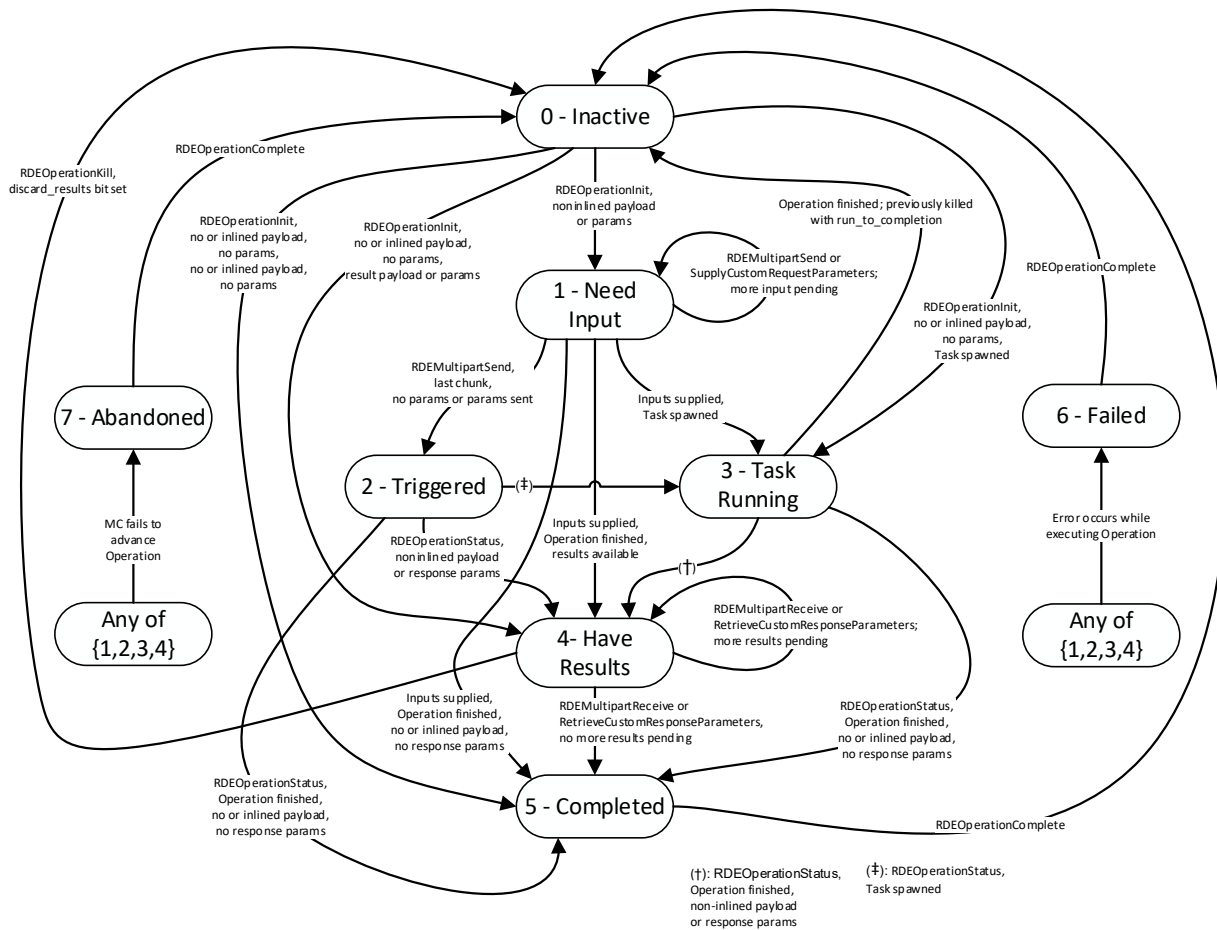
Current State	Trigger	Response	Next State
	RDEOperationKill <ul style="list-style-type: none"> - Operation executing - Operation can be killed - run_to_completion flag not set - discard_record flag set 	Success	INACTIVE
	RDEOperationKill <ul style="list-style-type: none"> - Operation executing - Operation can be killed - both run_to_completion and discard_record flags set 	Success	TASK_RUNNING
	RDEOperationKill; <ul style="list-style-type: none"> - Operation cannot be aborted or has finished execution - any combination of run_to_completion and discard_record flags set 	ERROR_OPERATION_UNKILLABLE	TASK_RUNNING
	Execution finishes; <ul style="list-style-type: none"> - Operation not killed 	Generate Task Completion Event (only once per Operation). Send to MC via PlatformEventMessage if MC configured the RDE Device to use asynchronous Events via SetEventReceiver; otherwise, MC will retrieve Event via PollForPlatformEventMessage. See Event lifecycle in clause 9.3 for further details	TASK_RUNNING
	Execution finishes; <ul style="list-style-type: none"> - Operation killed 	None	INACTIVE
	Execution finished; <ul style="list-style-type: none"> - Task Completion Event received by MC; - T_{abandon} timeout waiting for RDEOperationStatus command 	None	ABANDONED
	RDEOperationStatus; <ul style="list-style-type: none"> - execution not yet finished 	OPERATION_TASK_RUNNING	TASK_RUNNING
	RDEOperationStatus; <ul style="list-style-type: none"> - execution finished - payload too large for RDE Device to process 	ERROR_INSUFFICIENT_STORAGE	FAILED
	RDEOperationStatus; <ul style="list-style-type: none"> - execution finished - payload to be retrieved from RDE Device or response parameters present 	OPERATION_HAVE_RESULTS	HAVE_RESULTS

Current State	Trigger	Response	Next State
	RDEOperationStatus; <ul style="list-style-type: none"> - execution finished - no payload or payload fits in response message such that total response message size is within negotiated maximum transfer chunk - no response parameters 	OPERATION_COMPLETED	COMPLETED
	RDEMultipartSend, RDEMultipartReceive, RDEOperationComplete	ERROR_UNEXPECTED	TASK_RUNNING
	Any other Operation command	ERROR	TASK_RUNNING
4 - HAVE_RESULTS	RDEOperationInit, same rdeOpID	ERROR_OPERATION_EXISTS; no disruption to existing Operation	HAVE_RESULTS
	RDEOperationInit, different rdeOpID	Success or ERROR_CANNOT_CREATE_OPERATION, depending on whether the RDE Device has another slot to execute an Operation	The new Operation is tracked in a separate copy of the state machine; this Operation remains in HAVE_RESULTS
	RDEOperationKill <ul style="list-style-type: none"> - discard_results flag set - any other flag set 	ERROR_INVALID_DATA	HAVE_RESULTS
	RDEOperationKill <ul style="list-style-type: none"> - discard_results flag set - no other flag set 	SUCCESS	INACTIVE
	RDEOperationKill <ul style="list-style-type: none"> - run_to_completion flag set - discard_record flag not set 	ERROR_INVALID_DATA	HAVE_RESULTS
	RDEOperationKill; <ul style="list-style-type: none"> - any other combination of run_to_completion and discard_record flags set 	ERROR_OPERATION_UNKILLABLE	HAVE_RESULTS
	RDEOperationStatus	OPERATION_HAVE_RESULTS	HAVE_RESULTS
	RDEMultipartReceive; <ul style="list-style-type: none"> - MC aborts transfer 	Do not send data; Success; Prepare to restart transfer with next RDEMultipartReceive command	HAVE_RESULTS
	RDEMultipartReceive; <ul style="list-style-type: none"> - transfer error 	Error specific to type of transfer failure encountered	HAVE_RESULTS (MC may retry receive or abandon Operation)

Current State	Trigger	Response	Next State
	RDEMultipartReceive; <ul style="list-style-type: none"> - more data to transfer from the RDE Device to the MC after this chunk 	Send data; Success	HAVE_RESULTS
	RDEMultipartReceive; <ul style="list-style-type: none"> - no more data to transfer from the RDE Device to the MC after this chunk - response parameters to send 	Send data; Success	HAVE_RESULTS
	RDEMultipartReceive; <ul style="list-style-type: none"> - no more data to transfer from the RDE Device to the MC after this chunk - no response parameters present 	Send data; Success	COMPLETED
	T _{abandon} timeout waiting for RDEMultipartReceive and/or RetrieveCustomResponseParameters commands (depending on type of results still to be retrieved)	None	ABANDONED
	ReceiveCustomResponseParameters <ul style="list-style-type: none"> - RDE Device was not ready when RDEOperationInit command was sent and wished to specify a deferral timeframe 	Deferral Timeframe; Success	FAILED
	ReceiveCustomResponseParameters <ul style="list-style-type: none"> - response payload data not yet transferred 	Success	HAVE_RESULTS
	ReceiveCustomResponseParameters <ul style="list-style-type: none"> - response payload data partially transferred 	ERROR_UNEXPECTED	HAVE_RESULTS
	ReceiveCustomResponseParameters <ul style="list-style-type: none"> - no response payload or all response payload data transferred 	Success	COMPLETED
	Any other Operation or transfer command	Error	HAVE_RESULTS
5 - COMPLETED	RDEOperationInit, same rdeOpID	ERROR_OPERATION_EXISTS; no disruption to existing Operation	COMPLETED
	RDEOperationInit, different rdeOpID	Success or ERROR_CANNOT_CREATE_OPERATION, depending on whether the RDE Device has another slot to execute an Operation	The new Operation is tracked in a separate copy of the state machine; this

Current State	Trigger	Response	Next State
			Operation remains in COMPLETED
	RDEOperationKill - discard_results flag set - any other flag set	ERROR_INVALID_DATA	COMPLETED
	RDEOperationKill - discard_results flag set - no other flag set	ERROR_UNEXPECTED	COMPLETED
	RDEOperationKill - run_to_completion flag set - discard_record flag not set	ERROR_INVALID_DATA	COMPLETED
	RDEOperationKill; - any other combination of run_to_completion and discard_record flags set	ERROR_OPERATION_UNKILLABLE	COMPLETED
	RDEOperationStatus	OPERATION_COMPLETED	COMPLETED
	RDEOperationComplete	Success	INACTIVE
	Any other Operation command	Error	COMPLETED
6 - FAILED	RDEOperationInit, same rdeOpID	ERROR_OPERATION_EXISTS Operation	FAILED
	RDEOperationInit, different rdeOpID	Success or ERROR_CANNOT_CREATE_OPERATION, depending on whether the RDE Device has another slot to execute an Operation	The new Operation is tracked in a separate copy of the state machine; this Operation remains in FAILED
	RDEOperationKill - discard_results flag set - any other flag set	ERROR_INVALID_DATA	FAILED
	RDEOperationKill - discard_results flag set - no other flag set	ERROR_UNEXPECTED	FAILED
	RDEOperationKill - run_to_completion flag set - discard_record flag not set	ERROR_INVALID_DATA	FAILED
	RDEOperationKill - any other combination of run_to_completion and discard_record flags set	ERROR_OPERATION_FAILED	FAILED
	RDEOperationStatus	OPERATION_FAILED	FAILED
	RDEOperationComplete	Success	INACTIVE
	Any other Operation command	ERROR_OPERATION_FAILED	FAILED

Current State	Trigger	Response	Next State
7 - ABANDONED	RDEOperationInit, same rdeOpID	ERROR_OPERATION_EXISTS Operation	ABANDONED
	RDEOperationInit, different rdeOpID	Success or ERROR_CANNOT_CREATE_OP ERATION, depending on whether the RDE Device has another slot to execute an Operation	The new Operation is tracked in a separate copy of the state machine; this Operation remains in ABANDONED
	RDEOperationKill - discard_results flag set - any other flag set	ERROR_INVALID_DATA	ABANDONED
	RDEOperationKill - discard_results flag set - no other flag set	ERROR_UNEXPECTED	ABANDONED
	RDEOperationKill - run_to_completion flag set - discard_record flag not set	ERROR_INVALID_DATA	ABANDONED
	RDEOperationKill; - any other combination of run_to_completion and discard_record flags set	ERROR_OPERATION_ABANDONED	ABANDONED
	RDEOperationStatus	OPERATION_ABANDONED	ABANDONED
	RDEOperationComplete	Success	INACTIVE
	Any other Operation command	ERROR_OPERATION_ABANDONED	ABANDONED



2674

2675

Figure 16 – Operation lifecycle state machine (RDE Device perspective)

2676

9.3 Event lifecycle

2677

Table 50 describes the operating behavior for MCs and RDE Devices over the lifecycle of Events depicted visually in Figure 17. This sequence applies to both Task completion Events and schema-based Events. MC and RDE Device implementations of RDE shall comply with the sequences presented here.

2678

2679

2680

Table 50 – Event lifecycle overview

Current State	Description	Condition	Next Step
1 – OCCURS	The lifecycle of an Event begins when the Event occurs.	Unconditional	2
2 – RECORD	The RDE Device creates an Event record.	Unconditional	3
3 – ASYNC_CHK	The MC used the SetEventReceiver command to configure the RDE Device either to use asynchronous Events or to be polled for Events.	Asynchronous Events	6
		Polling	4
4 – EVT_POLL	The MC polls for Events using the PollForPlatformEventMessage command and discovers the Event.	Unconditional	5
5 – DISC_PREV	If the PollForPlatformEventMessage command request message reflected a previous Event to	Unconditional	8

Current State	Description	Condition	Next Step
	be acknowledged, the RDE Device discards the record for that previous Event.		
6 – EVT_SEND	The RDE Device issues a PlatformEventMessage command to the MC to notify it of the Event.	MC acknowledges the Event	7
		MC does not acknowledge the Event and retry count (PN1, see DSP0240) not exceeded	6
		MC does not acknowledge the Event and retry count exceeded	7
7 – DISC_RCRD	The RDE Device discards its Event record.	Unconditional	8
8 – MORE_CHK	Are there more Events (in the asynchronous case) or there was an Event to acknowledge (in the synchronous case)?	Yes	3
		No	9
9 – DONE	Event processing is complete.	n/a	-

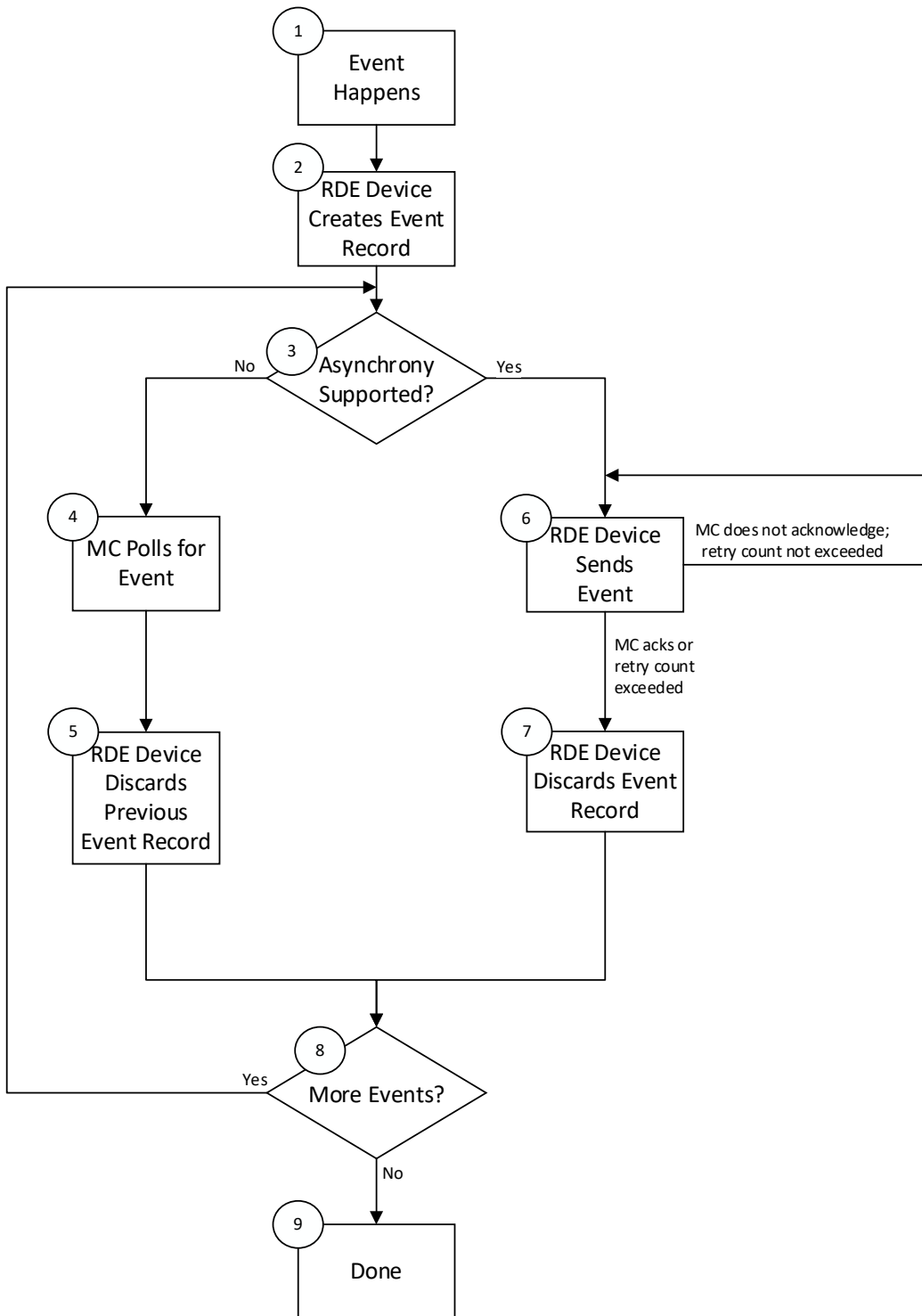


Figure 17 – Redfish event lifecycle overview

2681

2682

2683

10 PLDM commands for Redfish Device Enablement

2684

This clause provides the list of command codes that are used by MCs and RDE Devices that implement PLDM Redfish Device Enablement as defined in this specification. The command codes for the PLDM

2685

2686 messages are given in Table 51. RDE Devices and MCs shall implement all commands where the entry
 2687 in the “Command Requirement for RDE Device” or “Command Requirement for MC”, respectively, is
 2688 listed as Mandatory. RDE Devices and MCs may optionally implement any commands where the entry in
 2689 the “Command Requirement for RDE Device” or “Command Requirement for MC”, respectively, is listed
 2690 as Optional.

2691 **Table 51 – PLDM for Redfish Device Enablement command codes**

Command	Command Code	Command Requirement for RDE Device	Command Requirement for MC	Command Requestor (Initiator)	Reference
Discovery and Schema Management Commands					
NegotiateRedfishParameters	0x01	Mandatory	Mandatory	MC	See 11.1
NegotiateMediumParameters	0x02	Mandatory	Mandatory	MC	See 11.2
GetSchemaDictionary	0x03	Mandatory	Mandatory	MC	See 11.3
GetSchemaURI	0x04	Mandatory	Mandatory	MC	See 11.4
GetResourceETag	0x05	Mandatory	Mandatory	MC	See 11.5
GetOEMCount	0x06	Optional	Optional	MC	See 11.6
GetOEMName	0x07	Optional	Optional	MC	See 11.7
GetRegistryCount	0x08	Optional	Optional	MC	See 11.8
GetRegistryDetails	0x09	Optional	Optional	MC	See 11.9
SelectRegistryVersion	0x0A	Optional	Optional	MC	See 11.10
GetMessageRegistry	0x0B	Optional	Optional	MC	See 11.11
GetSchemaFile	0x0C	Optional	Optional	MC	See 11.12
Reserved	0x0D-0x0F				
RDE Operation and Task Commands					
RDEOperationInit	0x10	Mandatory	Mandatory	MC	See 12.1
SupplyCustomRequestParameters	0x11	Mandatory	Mandatory	MC	See 12.2
RetrieveCustomResponseParameters	0x12	Conditional ₄	Mandatory	MC	See 12.3
RDEOperationComplete	0x13	Mandatory	Mandatory	MC	See 12.4
RDEOperationStatus	0x14	Mandatory	Mandatory	MC	See 12.5
RDEOperationKill	0x15	Optional	Optional	MC	See 12.6
RDEOperationEnumerate	0x16	Mandatory	Optional	MC	See 12.7
Reserved	0x17-0x2F				
Multipart Transfer Commands					
RDEMultipartSend	0x30	Conditional ₁	Conditional ₁	MC	See 13.1
RDEMultipartReceive	0x31	Mandatory	Mandatory	MC	See 13.2
Reserved	0x32-0x3F				
Reserved For Future Use					
Reserved	0x40-0xFF				
Referenced PLDM Base Commands (PLDM Type 0)					

Command	Command Code	Command Requirement for RDE Device	Command Requirement for MC	Command Requestor (Initiator)	Reference
NegotiateTransferSize	See DSP0240	Conditional ₁	Conditional ₁	MC	See DSP0240
MultipartSend	See DSP0240	Conditional ₁	Conditional ₁	MC	See DSP0240
MultipartReceive	See DSP0240	Conditional ₁	Conditional ₁	MC	See DSP0240

Command	Command Code	Command Requirement for RDE Device	Command Requirement for MC	Command Requestor (Initiator)	Reference
Referenced PLDM for Monitoring and Control Commands (PLDM Type 2)					
GetPDRRepositoryInfo	See DSP0248	Mandatory	Mandatory	MC	See DSP0248
GetPDR	See DSP0248	Mandatory	Mandatory	MC	See DSP0248
SetEventReceiver	See DSP0248	Conditional ₂	Conditional ₂	MC	See DSP0248
PlatformEventMessage	See DSP0248	Optional ₃	Conditional ₃	RDE Device	See DSP0248
PollForPlatformEventMessage	See DSP0248	Optional ₂	Conditional ₃	MC	See DSP0248

2692 Notes:

- 2693 1) Either RDEMultipartSend or PLDM common MultipartSend is required if the RDE Device intends to
2694 support write Operations. RDE versions of bulk transfer commands shall be used if either the RDE
2695 Device or the MC does not support PLDM common versions; if both the RDE Device and the MC
2696 advertise support for PLDM common versions of bulk transfer commands (via the PLDM Base
2697 NegotiateTransferSize command), the RDE versions shall not be used.
- 2698 2) SetEventReceiver is mandatory if the RDE Device intends to support asynchronous messaging for
2699 Events via PlatformEventMessage.
- 2700 3) RDE Devices and MCs must support either PlatformEventMessage or
2701 PollForPlatformEventMessage in order to enable Event support.
- 2702 4) SupplyCustomResponseParameter is required if the RDE Device ever sets the
2703 HaveCustomResponseParameters flag in the OperationExecutionFlags field of the response
2704 message for a triggering command.

2705 **11 PLDM for Redfish Device Enablement – Discovery and schema**
2706 **commands**

2707 This clause describes the commands that are used by RDE Devices and MCs that implement the
2708 discovery and schema management commands defined in this specification. The command codes for the
2709 PLDM messages are given in Table 51.

2710 **11.1 NegotiateRedfishParameters command (0x01) format**

2711 This command enables the MC to negotiate general Redfish parameters with an RDE Device. The MC
2712 shall send this command to the RDE Device prior to any other RDE command. An RDE Device that
2713 supports multiple mediums shall provide the same response to this command independent of the medium
2714 on which this command was issued.

2715 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
2716 respond with data formatted per the Response Data section. For a non-SUCCESS CompletionCode, only
2717 the CompletionCode field of the Response Data shall be returned.

2718

Table 52 – NegotiateRedfishParameters command format

Type	Request data
uint8	<p>MCConcurrencySupport</p> <p>The maximum number of concurrent outstanding Operations the MC can support for this RDE Device. Must be > 0; a value of 1 indicates no support for concurrency. A value of 255 (0xFF) shall be interpreted to indicate that no such limit exists. Upon completion of this command, the RDE Device shall not initiate an Operation if MCConcurrencySupport (or DeviceConcurrencySupport whichever is lower) Operations are already active.</p>
bitfield16	<p>MCFeatureSupport</p> <p>Operations and functionality supported by the MC; for each, 1b indicates supported, 0b not:</p> <p>[15:9] - reserved</p> <p>[8] - BEJ v1.1 encoding and decoding supported; 1b = yes</p> <p>[7] - events_supported; 1b = yes. Must be 1b if MC supports Redfish Events or Long-running Tasks.</p> <p>[6] - action_supported; 1b = yes</p> <p>[5] - replace_supported; 1b = yes</p> <p>[4] - update_supported; 1b = yes</p> <p>[3] - delete_supported; 1b = yes</p> <p>[2] - create_supported; 1b = yes</p> <p>[1] - read_supported; 1b = yes. All MCs that implement PLDM for Redfish Device Enablement shall support read Operations</p> <p>[0] - head_supported; 1b = yes</p>
Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES }</p>
uint8	<p>DeviceConcurrencySupport</p> <p>The maximum number of concurrent outstanding Operations the RDE Device can support. Must be > 0; a value of 1 indicates no support for concurrency. A value of 255 (0xFF) shall be interpreted to indicate that no such limit exists. Regardless of the RDE Device’s level of support for concurrency, it shall not initiate an Operation if a limit indicated by MCConcurrencySupport has already been reached.</p>
bitfield8	<p>DeviceCapabilitiesFlags</p> <p>Capabilities for this RDE Device; for each, 1b indicates the RDE Device has the capability, 0b not:</p> <p>[7:3] - reserved</p> <p>[2] - bej_1_1_support: the RDE Device supported encoding and decoding BEJ version 1.1</p> <p>[1] - expand_support: the RDE Device can process a \$expand request query parameter (expressed via the LinkExpand field of the SupplyCustomRequestParameters command)</p> <p>[0] - atomic_resource_read: the RDE Device can respond to a read of an entire resource atomically, guaranteeing consistency of the read</p>

Type	Response data (continued)
bitfield16	<p>DeviceFeatureSupport</p> <p>Operations and functionality supported by this RDE Device; for each, 1b indicates supported, 0b not:</p> <p>[15:8] - reserved</p> <p>[7] - events_supported; 1b = yes. Must be 1b if RDE Device supports Redfish Events or Long-running Tasks. Shall match PLDM Event support indicated via support for PLDM for Platform Monitoring and Control (DSP0248) SetEventReceiver command</p> <p>[6] - action_supported; 1b = yes</p> <p>[5] - replace_supported; 1b = yes</p> <p>[4] - update_supported; 1b = yes</p> <p>[3] - delete_supported; 1b = yes</p> <p>[2] - create_supported; 1b = yes</p> <p>[1] - read_supported; 1b = yes. All RDE Devices shall support read Operations</p> <p>[0] - head_supported; 1b = yes</p>
uint32	<p>DeviceConfigurationSignature</p> <p>A signature (such as a CRC-32) calculated across all RDE PDRs and dictionaries that the RDE Device supports. This calculation should be performed as if all of the RDE PDRs and dictionaries were concatenated together into a single block of memory. The RDE Device may order the RDE PDRs and dictionaries in any sequence it chooses; however, it should be consistent in this ordering across invocations of the NegotiateRedfishParameters command. The RDE Device may use any method to generate the signature so long as it guarantees that a change to one or more RDE PDRs and/or dictionaries will not result in the same signature being generated.</p> <p>The RDE Device may generate the signature in any manner it sees fit; however, the signature generated for any given set of PDRs and dictionaries shall match any previous signature generated for the same set of PDRs and dictionaries. If a nonzero result from an RDE Device signature matches the result from a previous invocation of this command, the MC may generally assume that any RDE PDRs and/or dictionaries it has stored for the RDE Device remain unchanged and can be reused. However, MCs must be aware that any hashing algorithm risks a false positive match in result between hashes of two distinct sets of data. To mitigate this risk, MCs should utilize a secondary check, such as comparing the updateTime field in the PLDM for Platform Monitoring and Control GetPDRRepositoryInfo command response message to that from when PDRs were previously retrieved.</p>
varstring	<p>DeviceProviderName</p> <p>An informal name for the RDE Device</p>

2719 **11.2 NegotiateMediumParameters command (0x02) format**

2720 This command enables the MC to negotiate medium-specific parameters with an RDE Device. The MC
 2721 should invoke this command on each communication medium (e.g., RBT, SMBus, PCIe VDM) on which it
 2722 intends to interface with the RDE Device. The MC shall send this command over the transport for a
 2723 particular medium to negotiate parameters for that medium. When the RDE Device receives a request
 2724 with data formatted per the Request Data section below, it shall respond with data formatted per the
 2725 Response Data section. For a non-SUCCESS CompletionCode, only the CompletionCode field of the
 2726 Response Data shall be returned.

2727

Table 53 – NegotiateMediumParameters command format

Type	Request data
uint32	<p>MCMaximumTransferChunkSizeBytes</p> <p>An indication of the maximum amount of data the MC can support for a single message transfer. This value represents the size of the PLDM header and PLDM payload; medium specific header information shall not be included in this calculation. For cases of larger messages, a protocol-specific multipart transfer shall be utilized.</p> <p>All MC implementations shall support a transfer size of at least 64 bytes.</p> <p>NOTE For MCTP-based mediums, this is relative to the message size, not the packet size.</p>
Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES }</p> <p>If the MC reports a maximum transfer size of less than 64 bytes, the RDE Device shall respond with completion code ERROR_INVALID_DATA.</p>
uint32	<p>DeviceMaximumTransferChunkSizeBytes</p> <p>The maximum number of bytes that the RDE Device can support in a chunk for a single message transfer. This value represents the size of the PLDM header and PLDM payload; medium specific header information shall not be included in this calculation. If this value is greater than MCMaximumTransferChunkSizeBytes, the RDE Device shall “throttle down” to using the smaller value. If this value is smaller, the MC shall not attempt a transfer exceeding it.</p> <p>All RDE Device implementations shall support a transfer size of at least 64 bytes.</p> <p>NOTE For MCTP-based mediums, this is relative to the message size, not the packet size.</p>

2728 **11.3 GetSchemaDictionary command (0x03) format**

2729 This command enables the MC to retrieve a dictionary (full or truncated; see clause 7.2.2.3) associated
 2730 with a Redfish Resource PDR. After invoking the GetSchemaDictionary command, the MC shall, upon
 2731 receipt of a successful completion code and a valid read transfer handle, invoke one or more
 2732 RDEMultipartReceive commands (clause 13.2) to transfer data for the dictionary from the RDE Device.
 2733 The MC shall only have one dictionary, schema, or message registry retrieval in process from a given
 2734 RDE Device at any time. In the event that the MC begins a dictionary, schema, or message registry
 2735 retrieval when a previous retrieval has not yet completed (i.e., more chunks of dictionary or schema data
 2736 remain to be retrieved), the previous retrieval is implicitly aborted and the RDE Device may discard any
 2737 data associated with the transfer.

2738 MCs are discouraged from invoking the GetSchemaDictionary command in the middle of processing an
 2739 RDE Operation (excluding when it is running asynchronously as a long-running task). Instead, whenever
 2740 possible, they should run the Operation back to the INACTIVE state and only then retrieve dictionaries
 2741 needed to finalize processing of Operation results. (Ideally, these dictionaries would have been cached
 2742 before the Operation was initialized.) Neither the GetSchemaDictionary command nor any
 2743 RDEMultipartReceive commands used to retrieve a dictionary shall be construed as resetting the
 2744 abandonment timer ($T_{abandon}$, see clause 7.6).

2745 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2746 respond with data formatted per the Response Data section if it supports the command. For a non-
 2747 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2748

Table 54 – GetSchemaDictionary command format

Type	Request data
uint32	<p>ResourceID</p> <p>The ResourceID of any resource in the Redfish Resource PDR from which to retrieve the dictionary. A ResourceID of 0xFFFF FFFF may be supplied to retrieve dictionaries common to all RDE Device resources (such as the event or annotation dictionary) without referring to an individual resource.</p>
schemaClass	<p>RequestedSchemaClass</p> <p>The class of schema being requested</p>
Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES, ERROR_UNSUPPORTED, ERROR_NO_SUCH_RESOURCE }</p> <p>If the RDE Device does not support a schema of the type requested, it shall return CompletionCode ERROR_UNSUPPORTED. If the supplied Resource ID does not correspond to a collection, but the RequestedSchemaClass is COLLECTION_MEMBER_TYPE, the RDE Device shall return ERROR_INVALID_DATA.</p>
uint8	<p>DictionaryFormat</p> <p>The format of the dictionary as specified in the dictionary's VersionTag, defined in clause 7.2.3.2.</p>
uint32	<p>TransferHandle</p> <p>A data transfer handle that the MC shall use to retrieve the dictionary data via one or more RDEMultipartReceive commands (see clause 13.2). In conjunction with a non-failed CompletionCode, the RDE Device shall return a valid transfer handle.</p>

2749 **11.4 GetSchemaURI command (0x04) format**

2750 This command enables the MC to retrieve the formal URI for one of the RDE Device's schemas.

2751 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2752 respond with data formatted per the Response Data section if it supports the command. For a non-
 2753 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2754

Table 55 – GetSchemaURI command format

Type	Request data
uint32	ResourceID The ResourceID of a resource in a Redfish Resource PDR from which to retrieve the URI. A ResourceID of 0xFFFF FFFF may be supplied to retrieve URIs for schemas common to all RDE Device resources (such as for the annotation schema) without referring to an individual resource.
schemaClass	RequestedSchemaClass The class of schema being requested
uint8	OEMExtensionNumber Shall be zero for a standard DMTF-published schema, or the one-based OEM extension to a standard schema
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES, ERROR_UNSUPPORTED, ERROR_NO_SUCH_RESOURCE } For an out-of-range OEMExtensionNumber , the RDE Device shall return ERROR_INVALID_DATA. If the RDE Device does not support a schema of the type requested, it shall return CompletionCode ERROR_UNSUPPORTED.
uint8	StringFragmentCount The number of fragments N into which the URI string is broken; shall be greater than zero. The MC shall concatenate these together to reassemble the final string.
varstring	SchemaURI [0] URI string fragment for the schema. The reassembled string shall be the canonical URI for the JSON Schema used by the RDE Device.
...	...
varstring	SchemaURI [N - 1] URI string fragment for the schema. The reassembled string shall be the canonical URI for the JSON Schema used by the RDE Device.

2755 **11.5 GetResourceETag command (0x05) format**

2756 This command enables the MC to retrieve a hashed summary of the data contained immediately within a
 2757 resource, including all OEM extensions to it, or of all data within an RDE Device. The retrieved ETag shall
 2758 reflect the underlying data as specified in the Redfish specification ([DSP0266](#)).

2759 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2760 respond with data formatted per the Response Data section if it supports the command. For a non-
 2761 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2762 In the event that the RDE Device cannot provide a response to this command within the PT1 time period
 2763 (defined in [DSP0240](#)), the RDE Device may provide completion code ETAG_CALCULATION_ONGOING
 2764 and continue the process of generating the ETag. The MC may then poll for the completed ETag by
 2765 repeating the same GetResourceETag command that it gave that previously yielded this result. The RDE
 2766 Device in turn shall signal whether it has completed the calculation by responding with a completion code
 2767 of either SUCCESS (the calculation is done) or ETAG_CALCULATION_ONGOING (otherwise). It is
 2768 recommended that the MC delay for an integer multiple of PT1 between retry attempts.

2769 Following an invocation of this command that results in a completion code of
 2770 ETAG_CALCULATION_ONGOING, any other RDE command, including an invocation of
 2771 GetResourceETag with a different request message, shall be interpreted by the RDE Device as implicitly

2772 canceling the pending GetResourceETag command and cause it to stop generating the ETag. The RDE
 2773 Device shall then proceed to respond to the newly arrived command normally.

2774 NOTE ETags provided via this command are not escaped for inclusion in JSON data. MCs should be aware that
 2775 performing a raw comparison of an ETag retrieved from this command with one received as part of BEJ-encoded
 2776 JSON data will result in a mismatch as the ETag format requires characters that must be escaped in JSON data.

2777 **Table 56 – GetResourceETag command format**

Type	Request data
uint32	ResourceID The ResourceID of a resource in the Redfish Resource PDR for the instance from which to get an ETag digest; or 0xFFFF FFFF to get a global digest of all resource-based data within the RDE Device
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES, ERROR_NO_SUCH_RESOURCE, ETAG_CALCULATION_ONGOING }
varstring	ETag The RFC7232-compliant ETag string data; the string text format shall be UTF-8. Either a strong or a weak etag may be returned. This field shall be omitted if the CompletionCode is not SUCCESS.

2778 **11.6 GetOEMCount command (0x06) format**

2779 This command enables the MC to retrieve the number of OEM extensions for a schema.

2780 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2781 respond with data formatted per the Response Data section if it supports the command. For a non-
 2782 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2783 **Table 57 – GetOEMCount command format**

Type	Request data
uint32	ResourceID The ResourceID of the resource in the Redfish Resource PDR from which to retrieve the OEM count. A ResourceID of 0xFFFF FFFF may be supplied to retrieve OEM counts for schemas common to all RDE Device resources (such as the event dictionary) without referring to an individual resource.
schemaClasses	RequestedSchemaClass The class of schema being requested. NOTE Redfish does not allow OEM extensions to Annotation and Registry schemas.
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES, ERROR_NO_SUCH_RESOURCE }
uint8	OEMCount The number of OEM extensions associated with the schema. For schema classes that do not support OEM extensions this value shall be zero.

2784 **11.7 GetOEMName command (0x07) format**

2785 This command enables the MC to retrieve information about the name associated with an OEM extension
 2786 to a schema (including schemas for which OEM information is available in a Redfish Resource PDR).

2787 RDE Devices shall enumerate OEM extensions in lexicographic order.

2788 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2789 respond with data formatted per the Response Data section if it supports the command. For a non-
 2790 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2791 **Table 58 – GetOEMName command format**

Type	Request data
uint32	ResourceID The ResourceID of any resource in the Redfish Resource PDR from which to retrieve an OEM name. A ResourceID of 0xFFFF FFFF may be supplied to retrieve OEM names for extensions to schemas common to all RDE Device resources (such as the event dictionary) without referring to an individual resource.
schemaClass	RequestedSchemaClass The class of schema being requested
uint8	OEMIndex The zero-based index of the OEM extension about which information is to be retrieved. The total number of OEM extensions supported by an RDE Device for a given schema may be retrieved via the GetOEMCount command; the index supplied here should be less than that count.
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES, ERROR_NO_SUCH_RESOURCE } A response code of ERROR_INVALID_DATA shall be used to indicate when the supplied index does not exist in the schema or when the schema class does not support OEM schemas.
varstring	OEMName The OEM name associated with the extension

2792 **11.8 GetRegistryCount command (0x08) format**

2793 This command enables the MC to retrieve the number of message registries supported by an RDE
 2794 Device.

2795 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2796 respond with data formatted per the Response Data section if it supports the command. For a non-
 2797 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2798 **Table 59 – GetRegistryCount command format**

Type	Request data
--	None
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES }

uint8	RegistryCount The number of registries supported by the Device
-------	--------------------------------------------------------------------------

2799 **11.9 GetRegistryDetails command (0x09) format**

2800 This command enables the MC to retrieve information about a message registry an RDE Device supports.

2801 RDE Devices shall enumerate message registries in lexicographic order and return message registry
 2802 versions in reverse numeric order (most recent versions listed first). The RDE Device shall truncate the
 2803 list and decrease the count as needed to ensure that the response message fits within the negotiated
 2804 message size, thereby omitting mention of support for older versions.

2805 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2806 respond with data formatted per the Response Data section if it supports the command. For a non-
 2807 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2808 **Table 60 – GetRegistryDetails command format**

Type	Request data
uint8	RegistryIndex The zero-based index of the message registry about which information is to be retrieved. The total number of registries supported by an RDE Device may be retrieved via the GetRegistryCount command; the index supplied here should not exceed that count.
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES } ERROR_INVALID_DATA: The supplied index does not correspond to a supported registry
varstring	RegistryPrefix The Redfish prefix (name without version information) associated with the registry
varstring	RegistryURI URI at which the registry schema is published
uint8[2]	RegistryLanguage Language in which the registry is published, as an ISO 639-1 two-letter code
uint8	VersionCount The number N of registry versions the RDE Device supports for this registry
ver32	Version [0] First (newest) version of the registry supported
...	...
ver32	Version [N - 1] Last (oldest) version of the registry supported

2809 **11.10 SelectRegistryVersion command (0x0A) format**

2810 This command enables the MC to specify the version of a supported Redfish message registry that the
 2811 RDE device should use. By default, the RDE Device shall utilize the latest version of the registry that it
 2812 supports.

2813 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2814 respond with data formatted per the Response Data section if it supports the command.

2815

Table 61 – SelectRegistryVersion command format

Type	Request data
uint8	RegistryIndex The zero-based index of the message registry for which the registry is to be selected. The total number of registries supported by an RDE Device may be retrieved via the GetRegistryCount command; the index supplied here should be less than that count.
ver32	RegistryVersion Version of the registry to be used
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES } ERROR_INVALID_DATA: The supplied index does not correspond to a supported registry or the supplied version is not supported

2816 **11.11 GetMessageRegistry command (0x0B) format**

2817 This command enables the MC to retrieve the formal JSON registry for a Redfish message registry
 2818 supported by the RDE device. After invoking the GetMessageRegistry command, the MC shall, upon
 2819 receipt of a successful completion code and a valid read transfer handle, invoke one or more
 2820 RDEMultipartReceive commands (clause 13.2) to transfer data for the registry from the RDE Device. The
 2821 MC shall only have one dictionary, schema, or message registry retrieval in process from a given RDE
 2822 Device at any time. In the event that the MC begins a dictionary, schema, or message registry retrieval
 2823 when a previous retrieval has not yet completed (i.e., more chunks of dictionary or schema data remain to
 2824 be retrieved), the previous retrieval is implicitly aborted and the RDE Device may discard any data
 2825 associated with the transfer.

2826 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2827 respond with data formatted per the Response Data section if it supports the command. For a non-
 2828 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2829

Table 62 – GetMessageRegistry command format

Type	Request data
uint8	RegistryIndex The zero-based index of the message registry to be retrieved. The total number of registries supported by an RDE Device may be retrieved via the GetRegistryCount command; the index supplied here should not exceed that count.
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES } ERROR_INVALID_DATA: The supplied index does not correspond to a supported registry
uint8	SchemaFormat Bitwise OR of two values: Text format: { RAW_UTF8 = 0; GZIP_UTF8 = 1 } Schema format: { JSON = 0x10; CSDL = 0x20; YAML = 0x30 } In most cases, a message registry would be supplied as a GZIP'd UTF-8 JSON document, the value supplied would be 0x10.

uint32	<p>TransferHandle</p> <p>A data transfer handle that the MC shall use to retrieve the registry data via one or more RDEMultiPartReceive commands (see clause 13.2). In conjunction with a non-failed CompletionCode, the RDE Device shall return a valid transfer handle.</p>
--------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

2830 **11.12 GetSchemaFile command (0x0C) format**

2831 This command enables the MC to retrieve the formal schema for a Redfish resource supported by the
 2832 RDE device. After invoking the GetSchemaFile command, the MC shall, upon receipt of a successful
 2833 completion code and a valid read transfer handle, invoke one or more RDEMultiPartReceive commands
 2834 (clause 13.2) to transfer data for the schema from the RDE Device. The MC shall only have one
 2835 dictionary, schema, or message registry retrieval in process from a given RDE Device at any time. In the
 2836 event that the MC begins a dictionary, schema, or message registry retrieval when a previous retrieval
 2837 has not yet completed (i.e., more chunks of dictionary or schema data remain to be retrieved), the
 2838 previous retrieval is implicitly aborted and the RDE Device may discard any data associated with the
 2839 transfer. MCs should reference the version and signature of schemas, as documented in Redfish
 2840 Resource PDRs, wherever possible to avoid duplicate download of schema files.

2841 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2842 respond with data formatted per the Response Data section if it supports the command. For a non-
 2843 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2844 **Table 63 – GetSchemaFile command format**

Type	Request data
uint32	<p>ResourceID</p> <p>The ResourceID of a Redfish Resource PDR from which to retrieve the schema for an associated resource. A ResourceID of 0xFFFF FFFF may be supplied to retrieve a schema common to all RDE Device resources (such as the event or annotation dictionary) without referring to an individual resource.</p>
schemaClass	<p>RequestedSchemaClass</p> <p>The class of schema being requested</p>
uint8	<p>OEMOffset</p> <p>The offset for an OEM extension schema (see 11.6). A value of 0xFF shall be interpreted as requesting the base (standard) schema, including for schemas that do not support OEM extensions.</p>
Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES, ERROR_NO_SUCH_RESOURCE } ERROR_INVALID_DATA: The supplied OEMOffset is not valid</p>
uint8	<p>SchemaFormat</p> <p>Bitwise OR of two values: Text format: { RAW_UTF8 = 0; GZIP_UTF8 = 1 } Schema format: { JSON = 0x10; CSDL = 0x20; YAML = 0x30 } For example, for a CSDL (XML) format schema supplied as GZIP'd UTF-8 text, the value supplied would be 0x21.</p>
uint32	<p>TransferHandle</p> <p>A data transfer handle that the MC shall use to retrieve the registry data via one or more RDEMultiPartReceive commands (see clause 13.2). In conjunction with a non-failed CompletionCode, the RDE Device shall return a valid transfer handle.</p>

2845 **12 PLDM for Redfish Device Enablement – RDE Operation and Task**
 2846 **commands**

2847 This clause describes the Task commands that are used by RDE Devices and MCs that implement
 2848 Redfish Device Enablement as defined in this specification. The command numbers for the PLDM
 2849 messages are given in Table 51.

2850 **12.1 RDEOperationInit command (0x10) format**

2851 This command enables the MC to initiate a Redfish Operation with an RDE Device on behalf of a client.
 2852 After invoking the RDEOperationInit command, the MC may, upon receipt of a successful completion
 2853 code, invoke one or more RDEMultipartSend commands (clause 13.1) to transfer payload data of type
 2854 bejEncoding to the RDE Device. The MC shall only use RDEMultipartSend to transfer the payload data if
 2855 that data cannot fit in the request message of the RDEOperationInit command. After any payload has
 2856 been transferred, the MC may invoke the SupplyCustomRequestParameters command if additional
 2857 parameters are required. See clause 9 for more details on the Operation lifecycle.

2858 After the RDE Device receives the RDEOperationInit command, if flags are not set to indicate that it
 2859 should expect either payload data or custom request parameters, the RDE Device is triggered and shall
 2860 begin execution of the Operation. Similarly, if the flags are set to expect a payload but not parameters,
 2861 and the payload is contained inline in the request message, the RDE Device is implicitly triggered and
 2862 shall begin execution of the Operation.

2863 If triggered, the RDE Device shall respond with results if it is able to complete the Operation within the
 2864 time period required for a response to this message. If there is a response payload that fits within the
 2865 ResponsePayload field while maintaining a message size compatible with the negotiated maximum chunk
 2866 size (see NegotiateMediumParameters, clause 11.2), the RDE Device shall include it within this
 2867 response. Only if including a response payload would cause the message to exceed the negotiated chunk
 2868 size may the RDE Device flag it for transfer via RDEMultipartReceive.

2869 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2870 respond with data formatted per the Response Data section. Even with a non-SUCCESS
 2871 CompletionCode, all fields of the Response Data shall be returned.

2872 **Table 64 – RDEOperationInit command format**

Type	Request data
uint32	ResourceID The resourceID of a resource in the Redfish Resource PDR for the data that is the target of this operation
rdeOpID	OperationID Identification number for this Operation; must match the one used for all commands relating to this Operation. NOTE Operation IDs with the most significant bit cleared are reserved for use by the RDE Device; it is an error for the MC to supply such an ID.
enum8	OperationType The type of Redfish Operation being performed. values: { OPERATION_HEAD = 0; OPERATION_READ = 1; OPERATION_CREATE = 2; OPERATION_DELETE = 3; OPERATION_UPDATE = 4; OPERATION_REPLACE = 5; OPERATION_ACTION = 6 }

bitfield8	<p>OperationFlags</p> <p>Flags associated with this Operation:</p> <p>[7:4] - reserved for future use</p> <p>[3] - excerpt_flag; if 1b, the RDE Device should perform an excerpt read (see 7.2.3.11.6)</p> <p>[2] - contains_custom_request_parameters; if 1b, the RDE Device should expect to receive a SupplyCustomRequestParameters command request before it may trigger the Operation</p> <p>[1] - contains_request_payload; if 0b, the Operation does not require data to be sent</p> <p>[0] - locator_valid; if 0b, the locator in the OperationLocator field shall be ignored</p>
uint32	<p>SendDataTransferHandle</p> <p>Handle to be used with the first RDEMultiPartSend command transferring BEJ formatted data for the operation. If no data is to be sent for this operation or if the request payload fits entirely within this request message, then it shall be zero (0x00000000) (see the RequestPayloadLength and RequestPayload fields below).</p>
uint8	<p>OperationLocatorLength</p> <p>Length in bytes of the OperationLocator for this Operation. This field shall be zero (0x00) if the locator_valid bit in the OperationFlags field above is set to 0b or if the OperationType field above is not one of OPERATION_UPDATE and OPERATION_ACTION.</p>
uint32	<p>RequestPayloadLength</p> <p>Length in bytes of the request payload in this message. This value shall be zero (0x00000000) under either of the following conditions:</p> <ul style="list-style-type: none"> • There is no request payload as indicated by contains_request_payload bit of the OperationFlags parameter above • The entire payload cannot fit within this message, subject to the maximum transfer chunk size as determined at registration time via the NegotiateMediumParameters command
bejLocator	<p>OperationLocator</p> <p>BEJ locator indicating where the new Operation is to take place within the resource specified in ResourceID.</p> <p>When the OperationType is set to OPERATION_ACTION, this field shall be set to the location of an action within a particular resource dictionary.</p> <p>A BEJ locator shall not be set to the location of an action unless the OperationType is set to OPERATION_ACTION. Similarly, a BEJ locator shall never be set to the location of any of the fields within the action set. The RDE device shall treat a locator in violation of either of these rules as invalid and shall return ERROR_INVALID_DATA in this case.</p> <p>This field may not be supported for other operation types and shall be omitted if the OperationLocatorLength field above is set to zero.</p>
null or bejEncoding	<p>RequestPayload</p> <p>The request payload. The format of this parameter shall be null (consisting of zero bytes) if the RequestPayloadLength above is zero; it shall be bejEncoding otherwise.</p> <p>When the OperationType is set to OPERATION_ACTION, and the requested action contains parameters, the request payload shall consist of the set corresponding to the action; this set shall then in turn contain any parameters associated with the action.</p> <p>If the action does not require any parameters, the request payload shall be null.</p>

Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES, ERROR_CANNOT_CREATE_OPERATION, ERROR_NOT_ALLOWED, ERROR_WRONG_LOCATION_TYPE, ERROR_OPERATION_EXISTS, ERROR_UNSUPPORTED, ERROR_NO_SUCH_RESOURCE, ERROR_INSUFFICIENT_STORAGE }</p> <p>Response codes ERROR_CANNOT_CREATE_OPERATION, ERROR_NOT_ALLOWED, ERROR_WRONG_LOCATION_TYPE, ERROR_OPERATION_EXISTS, ERROR_UNSUPPORTED, ERROR_NO_SUCH_RESOURCE, and ERROR_INSUFFICIENT_STORAGE shall be interpreted to represent an operational failure, not a command failure.</p>
enum8	<p>OperationStatus</p> <p>values: { OPERATION_INACTIVE = 0; OPERATION_NEEDS_INPUT = 1; OPERATION_TRIGGERED = 2; OPERATION_RUNNING = 3; OPERATION_HAVE_RESULTS = 4; OPERATION_COMPLETED = 5, OPERATION_FAILED = 6, OPERATION_ABANDONED = 7 }</p>
uint8	<p>CompletionPercentage</p> <p>0..100: percentage complete; 101-253: reserved for future use; 254: not supported or otherwise unable to estimate (but a valid Operation) 255: invalid Operation</p> <p>This value shall be zero if the Operation has not yet been triggered or if the Operation has failed.</p>
uint32	<p>CompletionTimeSeconds</p> <p>An estimate of the number of seconds remaining before the Operation is completed, or 0xFFFF FFFF if such an estimate cannot be provided.</p> <p>This value shall be 0xFFFF FFFF if the Operation has not yet been triggered or if the Operation has failed.</p>
bitfield8	<p>OperationExecutionFlags</p> <p>[7:4] - Reserved</p> <p>[3] - CacheAllowed – 1b = yes; shall be 0b for Operations other than read, head. Shall be 0b unless Operation has finished. Referring to RFC 7234, a value of yes shall be considered as equivalent to Cache-Control response header value “public” and a value of no shall be considered as equivalent to Cache-Control response header value “no-store”. Other cache directives are not supported. The decision of whether to allow caching of data is up to the RDE Device. Typically, static data is allowed to be cached unless, for example, it represents sensitive data such as login credentials; data that changes over time is generally not marked as cacheable.</p> <p>To process the CacheAllowed flag, the MC shall behave as described in clause 7.2.3.10.7</p> <p>[2] - HaveResultPayload – 1b = yes. Shall be 0b if Operation has not finished</p> <p>[1] - HaveCustomResponseParameters – 1b = yes. Shall be 0b if Operation has not finished</p> <p>[0] - TaskSpawned – 1b = yes</p> <p>For a failed Operation, this field shall be 0b for all flags other than HaveResultPayload, which may be 1b if a @Message.ExtendedInfo annotation is available to explain the result.</p>
uint32	<p>ResultTransferHandle</p> <p>A data transfer handle that the MC may use to retrieve a larger response payload via one or more RDEMultipartReceive commands (see clause 13.2). The RDE Device shall return a transfer handle of 0xFFFFFFFF if Operation execution has not finished or if the Operation has not yet been triggered. In the event of a failed Operation, or if the data fits entirely within the payload of this command response, or if there is no data to retrieve, the RDE Device shall return a null transfer handle, 0x00000000.</p>

Type	Response data (continued)
bitfield8	<p>PermissionFlags</p> <p>Indicates the access level (types of Operations; see Table 33) granted to the resource targeted by the Operation.</p> <p>[7: 6] - reserved for future use</p> <p>[5] - head access; 1b = access allowed</p> <p>[4] - delete access; 1b = access allowed</p> <p>[3] - create access; 1b = access allowed</p> <p>[2] - replace access; 1b = access allowed</p> <p>[1] - update access; 1b = access allowed</p> <p>[0] - read access; 1b = access allowed</p> <p>Additional notes on processing PermissionFlags may be found in clause 7.2.3.10.8.</p>
uint32	<p>ResponsePayloadLength</p> <p>Length in bytes of the response payload in this message. This value shall be zero under any of the following conditions:</p> <ul style="list-style-type: none"> • The Operation has not yet been triggered. • The Operation status is not completed or failed, as indicated by the OperationStatus parameter above. For a failed Operation, a @Message.ExtendedInfo annotation may be supplied in the response payload. • There is no response payload as indicated by Bit 2 of the OperationExecutionFlags parameter above. • The entire payload cannot fit within this message, subject to the maximum transfer chunk size as determined at registration time via the NegotiateMediumParameters command.
varstring	<p>ETag</p> <p>String data for an ETag digest of the target resource; the string text format shall be UTF-8. The ETag shall be skipped (a string consisting of just the null terminator returned in this field) for any of the following actions: Action, Delete, Replace, and Update. The ETag shall also be skipped (a string consisting of just the null terminator returned in this field) if execution of the Operation has failed or not yet finished.</p> <p>Additional notes on processing ETags may be found in clause 7.2.3.10.4.</p> <p>NOTE ETags provided via this field are not escaped for inclusion in JSON data as they are primarily intended to be used for the ETag HTML header. MCs should be aware that performing a raw comparison of an ETag retrieved from this command with one received as part of BEJ-encoded JSON data will result in a mismatch as the ETag format requires characters that must be escaped in JSON data.</p>
null or bejEncoding	<p>ResponsePayload</p> <p>The response payload. The format of this parameter shall be null (consisting of zero bytes) if the ResponsePayloadLength above is zero; it shall be bejEncoding otherwise.</p>

2873 **12.2 SupplyCustomRequestParameters command (0x11) format**

2874 This command enables the MC to send custom HTTP/HTTPS X- headers and other uncommon request
 2875 parameters to an RDE Device to be applied to an Operation if the client's HTTP operation contains any
 2876 such parameters. The MC must not use this command to submit any headers for which a standard
 2877 handling is defined in either this specification or [DSP0266](#). If the client's HTTP operation does not contain
 2878 the parameters conveyed in this command, the MC shall not send this command as part of its processing
 2879 of the Operation.

2880 The MC shall only invoke this command in the event that at least one custom header or uncommon
 2881 request parameter needs to be transferred to the RDE Device. When sent, the
 2882 **SupplyCustomRequestParameters** command shall be invoked after the MC sends the
 2883 RDEOperationInit command.

2884 After the RDE Device receives the SupplyCustomRequestParameters command, if flags from the original
 2885 RDEOperationInit command (see clause 12.1) were not set to indicate that it should expect payload data
 2886 or if the RDE Device has already received payload data, the RDE Device shall consider itself triggered
 2887 and begin execution of the Operation.

2888 If triggered, the RDE Device shall respond with results if it is able to complete the Operation within the
 2889 time period required for a response to this message. If there is a response payload that fits within the
 2890 ResponsePayload field while maintaining a message size compatible with the negotiated maximum chunk
 2891 size (see clause 11.2), the RDE Device shall include it within this response. Only if including a response
 2892 payload would cause the message to exceed the negotiated chunk size may the RDE Device flag it for
 2893 transfer via RDEMultipartReceive.

2894 The size of the request message is limited to the negotiated maximum chunk size (see clause 11.2). If the
 2895 client supplied sufficiently many custom request headers and/or ETags that the request message would
 2896 exceed this negotiated size, the MC shall abort the request and perform the following steps:

- 2897 1) Use the RDEOperationKill (see clause 12.6) and then RDEOperationComplete (see clause
 2898 12.4) commands to abort and finalize the Operation if it had already been initiated via
 2899 RDEOperationInit (see clause 12.1).
- 2900 2) Return to the client HTTP/HTTPS error code 431, Request Header Fields Too Large.
- 2901 3) Cease processing of the client request.

2902 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2903 respond with data formatted per the Response Data section. Even with a non-SUCCESS
 2904 CompletionCode, all fields of the Response Data shall be returned.

2905 **Table 65 – SupplyCustomRequestParameters command format**

Type	Request data
uint32	ResourceID The resourceID of a resource in the Redfish Resource PDR for the instance to which custom headers should be supplied
rdeOpID	OperationID Identification number for this Operation; must match the one used for all commands relating to this Operation.
uint16	LinkExpand The value of a \$levels qualifier to a \$expand query option if supplied as part of an HTTP/HTTPS GET operation. The MC shall supply a value of zero if the query option was not supplied. This integer indicates the number of levels of links to expand when reading data from a resource. The MC shall supply a value of zero if the \$expand query option was not supplied. See DSP0266 for more details. This value should be ignored by the RDE Device if it did not set expand_support in the DeviceCapabilitiesFlags response parameter to the NegotiateRedfishParameters command. To process the LinkExpand parameters, the MC and RDE Device shall behave as described in clause 7.2.3.11.3. In particular, when supporting this command, an RDE Device shall encode pages expanded into with the bejResourceLinkExpansion format specification.

Type	Request data (continued)
uint16	<p>CollectionSkip</p> <p>The value of a \$skip query option if supplied as part of an HTTP/HTTPS GET operation. The MC shall supply a value of zero if the \$skip query option was not supplied. This integer indicates the number of Members in a resource collection to skip before retrieving the first resource. See DSP0266 for more details.</p> <p>Additional notes on processing the \$skip query option may be found in clause 7.2.3.11.1.</p>
uint16	<p>CollectionTop</p> <p>The value of a \$top query option if supplied as part of an HTTP/HTTPS GET operation. The MC shall supply a value of 0xFFFF (to be treated by the RDE Device as unlimited) if the query option was not supplied. This indicates the number of Members of a resource collection to include in a response. See DSP0266 for more details.</p> <p>Additional notes on processing the \$top query option may be found in clause 7.2.3.11.2.</p>
uint16	<p>PaginationOffset</p> <p>The page offset for paginated response data that the RDE Device supplied in conjunction with an @odata.nextlink annotation and decoded from a pagination URI. Shall be 0 if no pagination has taken place. See clause 14.2.8 for more details on RDE Device-selected dynamic pagination.</p> <p>Additional notes on pagination may be found in clause 14.2.8.</p>
enum8	<p>ETagOperation</p> <p>To process an ETagOperation, the RDE Device shall respond as described in clauses 7.2.3.10.1 and 7.2.3.10.2.</p> <p>values: { ETAG_IGNORE = 0; ETAG_IF_MATCH = 1; ETAG_IF_NONE_MATCH = 2 }</p>
uint8	<p>ETagCount</p> <p>Number of ETags supplied in this message; should be zero if ETagOperation above is ETAG_IGNORE and nonzero otherwise.</p>
varstring	<p>ETag [0]</p> <p>String data for first ETag, if ETagCount > 0. This string shall be UTF-8 format.</p> <p>Additional notes on processing ETags may be found in clause 7.2.3.10.4.</p>
...	Additional ETags
uint8	<p>HeaderCount</p> <p>The number of RDE custom headers being supplied in this operation.</p> <p>Additional notes on processing RDE custom headers may be found in clause 7.2.3.10.3.</p>
varstring	<p>HeaderName [0]</p> <p>The name of the header, including the PLDM-RDE- prefix</p>
varstring	<p>HeaderParameter [0]</p> <p>The parameter or parameters associated with the header. The MC may preprocess these – though any such preprocessing is outside the scope of this specification – or convey them exactly as received.</p>
...	...
Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES, ERROR_OPERATION_ABANDONED, ERROR_OPERATION_FAILED, ERROR_UNSUPPORTED, ERROR_UNEXPECTED, ERROR_UNRECOGNIZED_CUSTOM_HEADER, ERROR_ETAG_MATCH, ERROR_NO_SUCH_RESOURCE, ERROR_INSUFFICIENT_STORAGE }</p> <p>Response codes ERROR_UNSUPPORTED, ERROR_UNRECOGNIZED_CUSTOM_HEADER, and ERROR_INSUFFICIENT_STORAGE shall be used to indicate that an unsupported request parameter was sent. These responses represent an Operational failure, not a command failure.</p>

Type	Response data (continued)
enum8	<p>OperationStatus values: { OPERATION_INACTIVE = 0; OPERATION_NEEDS_INPUT = 1; OPERATION_TRIGGERED= 2; OPERATION_RUNNING = 3; OPERATION_HAVE_RESULTS = 4; OPERATION_COMPLETED = 5, OPERATION_FAILED = 6, OPERATION_ABANDONED = 7 }</p>
uint8	<p>CompletionPercentage 0..100: percentage complete; 101-253: reserved for future use; 254: not supported or otherwise unable to estimate (but a valid Operation) 255: invalid Operation This value shall be zero if the Operation has not yet been triggered or if the Operation has failed.</p>
uint32	<p>CompletionTimeSeconds An estimate of the number of seconds remaining before the Operation is completed, or 0xFFFF FFFF if such an estimate cannot be provided. This value shall be 0xFFFF FFFF if the Operation has not yet been triggered or if the Operation has failed.</p>
bitfield8	<p>OperationExecutionFlags [7:4] - Reserved [3] - CacheAllowed – 1b = yes; shall be 0b for Operations other than read, head. Shall be 0b unless Operation has finished. Referring to RFC 7234, a value of yes shall be considered as equivalent to Cache-Control response header value “public” and a value of no shall be considered as equivalent to Cache-Control response header value “no-store”. Other cache directives are not supported. The decision of whether to allow caching of data is up to the RDE Device. Typically, static data is allowed to be cached unless, for example, it represents sensitive data such as login credentials; data that changes over time is generally not marked as cacheable To process the CacheAllowed flag, the MC shall behave as described in clause 7.2.3.10.7 [2] - HaveResultPayload – 1b = yes. Shall be 0b if Operation has not finished [1] - HaveCustomResponseParameters – 1b = yes. Shall be 0b if Operation has not finished [0] - TaskSpawned – 1b = yes For a failed Operation, this field shall be 0b for all flags other than HaveResultPayload, which may be 1b if a @Message.ExtendedInfo annotation is available to explain the result.</p>
uint32	<p>ResultTransferHandle A data transfer handle that the MC may use to retrieve a larger response payload via one or more RDEMultipartReceive commands (see clause 13.2). The RDE Device shall return a transfer handle of 0xFFFFFFFF if Operation execution has not finished or if the Operation has not yet been triggered. In the event of a failed Operation, or if the data fits entirely within the payload of this command response, or if there is no data to retrieve, the RDE Device shall return a null transfer handle, 0x00000000.</p>

Type	Response data (continued)
bitfield8	<p>PermissionFlags</p> <p>Indicates the access level (types of Operations; see Table 33) granted to the resource targeted by the Operation.</p> <p>[7:6] - reserved for future use</p> <p>[5] - head access; 1b = access allowed</p> <p>[4] - delete access; 1b = access allowed</p> <p>[3] - create access; 1b = access allowed</p> <p>[2] - replace access; 1b = access allowed</p> <p>[1] - update access; 1b = access allowed</p> <p>[0] - read access; 1b = access allowed</p> <p>The MC and RDE Device shall process PermissionFlags as described in clause 7.2.3.10.8.NOTE: The bit mapping for the PermissionFlags field was changed in version 1.0.1 of this specification to match that from the RDEOperationInit command, thereby making the entire response message identical for both of these commands.</p>
uint32	<p>ResponsePayloadLength</p> <p>Length in bytes of the response payload in this message. This value shall be zero under any of the following conditions:</p> <ul style="list-style-type: none"> • The Operation has not yet been triggered • The Operation status is not completed or failed, as indicated by the OperationStatus parameter above. For a failed Operation, a @Message.ExtendedInfo annotation may be supplied in the response payload. • There is no response payload as indicated by Bit 2 of the OperationExecutionFlags parameter above • The entire payload cannot fit within this message, subject to the maximum transfer chunk size as determined at registration time via the NegotiateMediumParameters command
varstring	<p>ETag</p> <p>String data for an ETag digest of the target resource; the string text format shall be UTF-8. The ETag may be skipped (an empty string returned in this field) for any of the following actions: Action, Delete, Replace, and Update. The ETag shall also be skipped (an empty string returned in this field) if execution of the Operation has not yet finished.</p> <p>This field supports the ETag Response header. Additional notes on processing ETags may be found in clause 7.2.3.10.4.</p> <p>NOTE ETags provided via this field are not escaped for inclusion in JSON data as they are primarily intended to be used for the ETag HTML header. MCs should be aware that performing a raw comparison of an ETag retrieved from this command with one received as part of BEJ-encoded JSON data will result in a mismatch as the ETag format requires characters that must be escaped in JSON data.</p>
null or bejEncoding	<p>ResponsePayload</p> <p>The response payload. The format of this parameter shall be null (consisting of zero bytes) if the ResponsePayloadLength above is zero; it shall be bejEncoding otherwise.</p>

2906 **12.3 RetrieveCustomResponseParameters command (0x12) format**

2907 This command enables the MC to retrieve custom HTTP/HTTPS headers or other uncommon response
 2908 parameters from an RDE Device to be forwarded to the client that initiated a Redfish operation. The MC
 2909 shall only invoke this command when the **HaveCustomResponseParameters** flag in the response
 2910 message for a triggered RDE command indicates that it is needed.

2911 The RDE Device shall not supply more response headers than would allow the response message to fit in
 2912 the negotiated maximum transfer chunk size (see clause 11.2).

2913 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2914 respond with data formatted per the Response Data section. For a non-SUCCESS CompletionCode, only
 2915 the CompletionCode field of the Response Data shall be returned.

2916 **Table 66 – RetrieveCustomResponseParameters command format**

Type	Request data
uint32	ResourceID The resourceID of a resource in the Redfish Resource PDR for the instance from which custom headers should be reported
rdeOpID	OperationID Identification number for this Operation; must match the one used for all commands relating to this Operation
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES, ERROR_OPERATION_ABANDONED, ERROR_OPERATION_FAILED, ERROR_UNEXPECTED, ERROR_NO_SUCH_RESOURCE }
uint32	DeferralTimeframe The expected length of time in seconds before the RDE Device will be able to respond to a request to start an Operation, or 0xFF if unknown. The MC shall ignore this field except when the completion code of the previous RDEOperationInit was ERROR_NOT_READY. This field supports the Retry-After response header. Additional notes on processing the Retry-After response header may be found in clause 7.2.3.10.9.
uint32	NewResourceID Resource ID for a newly created collection entry; this value shall be 0 and ignored if the Operation is not a Redfish Create or if the Operation has failed or not yet completed. This field supports the Location Response header. Additional notes on processing the Location response header may be found in clause 7.2.3.10.6.
uint8	ResponseHeaderCount Number of custom response headers contained in the remainder of this message
varstring	HeaderName [0] The name of the header, including the X- prefix This field shall be omitted if ResponseHeaderCount above is zero
varstring	HeaderParameter [0] The parameter or parameters associated with the header. The MC may preprocess these – though any such preprocessing is outside the scope of this specification – or convey them exactly as received This field shall be omitted if ResponseHeaderCount above is zero
...	...

2917 **12.4 RDEOperationComplete command (0x13) format**

2918 This command enables the MC to inform an RDE Device that it considers an Operation to be complete,
 2919 including failed and abandoned Operations. The RDE Device in turn may discard any internal records for
 2920 the Operation.

2921 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2922 respond with data formatted per the Response Data section.

2923

Table 67 – RDEOperationComplete command format

Type	Request data
uint32	ResourceID The resourceID of a resource in the Redfish Resource PDR to which the Task’s operation was targeted
rdeOpID	OperationID Identification number for this Operation; must match the one used for all commands relating to this Operation
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES, ERROR_UNEXPECTED, ERROR_NO_SUCH_RESOURCE }

2924 **12.5 RDEOperationStatus command (0x14) format**

2925 This command enables the MC to query an RDE Device for the status of an Operation. It is additionally
 2926 used to collect the initial response when an RDE Operation is triggered by a RDEMultiPartSend command
 2927 or after a Task finishes asynchronous execution.

2928 When providing result data for an Operation that has finished executing, if there is a response payload
 2929 that fits within the ResponsePayload field while maintaining a message size compatible with the
 2930 negotiated maximum chunk size (see NegotiateMediumParameters, clause 11.2), the RDE Device shall
 2931 include it within this response. Only if including a response payload would cause the message to exceed
 2932 the negotiated chunk size may the RDE Device flag it for transfer via RDEMultiPartReceive.

2933 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2934 respond with data formatted per the Response Data section. Even with a non-SUCCESS
 2935 CompletionCode, all fields of the Response Data shall be returned.

2936

Table 68 – RDEOperationStatus command format

Type	Request data
uint32	ResourceID The resourceID of a resource in the Redfish Resource PDR to which the Task’s operation was targeted
rdeOpID	OperationID Identification number for this Operation; must match the one used for all commands relating to this Operation

Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES, ERROR_UNSUPPORTED, ERROR_ETAG_MATCH, ERROR_UNRECOGNIZED_CUSTOM_HEADER, ERROR_INSUFFICIENT_STORAGE }</p> <p>The completion code for RDEOperationStatus shall be one of the following:</p> <p>SUCCESS: An RDE Operation was referenced in the OperationID request field and it is not in the failed state. The actual current status of the RDE Operation is returned in the OperationStatus field. If the OperationID does not correspond to an active Operation, the state shall be reported as OPERATION_INACTIVE.</p> <p>ERROR_UNSUPPORTED, ERROR_ETAG_MATCH, ERROR_UNRECOGNIZED_CUSTOM_HEADER, ERROR_INSUFFICIENT_STORAGE : An RDE Operation in the FAILED state was referenced in the OperationID request field, and the Operation failed with the specified status code. OperationStatus shall be OPERATION_FAILED in this case. These responses indicate a failure in the RDE Operation, not a failure in the RDEOperationStatus command.</p>
enum8	<p>OperationStatus</p> <p>values: { OPERATION_INACTIVE = 0; OPERATION_NEEDS_INPUT = 1; OPERATION_TRIGGERED= 2; OPERATION_RUNNING = 3; OPERATION_HAVE_RESULTS = 4; OPERATION_COMPLETED = 5, OPERATION_FAILED = 6, OPERATION_ABANDONED = 7 }</p>
uint8	<p>CompletionPercentage</p> <p>0..100: percentage complete; 101-253: reserved for future use; 254: not supported or otherwise unable to estimate (but a valid Operation) 255: invalid Operation</p> <p>This value shall be zero if the Operation has not yet been triggered or if the Operation has failed.</p>
uint32	<p>CompletionTimeSeconds</p> <p>An estimate of the number of seconds remaining before the Operation is completed, or 0xFFFF FFFF if such an estimate cannot be provided.</p> <p>This value shall be 0xFFFF FFFF if the Operation has not yet been triggered or if the Operation has failed.</p>
bitfield8	<p>OperationExecutionFlags</p> <p>[7:4] - Reserved</p> <p>[3] - CacheAllowed – 1b = yes; shall be 0b for Operations other than read, head. Shall be 0b unless Operation has finished. Referring to RFC 7234, a value of yes shall be considered as equivalent to Cache-Control response header value “public” and a value of no shall be considered as equivalent to Cache-Control response header value “no-store”. Other cache directives are not supported. The decision of whether to allow caching of data is up to the RDE Device. Typically, static data is allowed to be cached unless, for example, it represents sensitive data such as login credentials; data that changes over time is generally not marked as cacheable</p> <p>To process the CacheAllowed flag, the MC shall behave as described in clause 7.2.3.10.7</p> <p>[2] - HaveResultPayload – 1b = yes. Shall be 0b if Operation has not finished</p> <p>[1] - HaveCustomResponseParameters – 1b = yes. Shall be 0b if Operation has not finished</p> <p>[0] - TaskSpawned – 1b = yes</p> <p>For a failed Operation, this field shall be 0b for all flags other than HaveResultPayload, which may be 1b if a @Message.ExtendedInfo annotation is available to explain the result.</p>

Type	Response data (continued)
uint32	<p>ResultTransferHandle</p> <p>A data transfer handle that the MC may use to retrieve a larger response payload via one or more RDEMultipartReceive commands (see clause 13.2). The RDE Device shall return a transfer handle of 0xFFFFFFFF if Operation execution has not finished or if the Operation has not yet been triggered. In the event of a failed Operation, or if the data fits entirely within the payload of this command response, or if there is no data to retrieve, the RDE Device shall return a null transfer handle, 0x00000000.</p> <p>In the event that data transfer for this Operation is currently in progress (at least one chunk has been transferred but the final chunk has not yet been transferred, and a timeout has not occurred awaiting the request for the next chunk), the RDE Device shall return the transfer handle that was most recently returned in the response message for a RDEMultipartSend or RDEMultipartReceive command.</p>
bitfield8	<p>PermissionFlags</p> <p>Indicates the access level (types of Operations; see Table 33) granted to the resource targeted by the Operation.</p> <p>[7:6] - reserved for future use</p> <p>[5] - head access; 1b = access allowed</p> <p>[4] - delete access; 1b = access allowed</p> <p>[3] - create access; 1b = access allowed</p> <p>[2] - replace access; 1b = access allowed</p> <p>[1] - update access; 1b = access allowed</p> <p>[0] - read access; 1b = access allowed</p> <p>This field supports the Allow header. Additional notes on processing the Allow header may be found in clause 7.2.3.10.8</p> <p>.NOTE: The bit mapping for the PermissionFlags field was changed in version 1.0.1 of this specification to match that from the RDEOperationInit command, thereby making the entire response message identical for both of these commands.</p>
uint32	<p>ResponsePayloadLength</p> <p>Length in bytes of the response payload in this message. This value shall be zero under any of the following conditions:</p> <ul style="list-style-type: none"> • The Operation has not yet been triggered • The Operation status is not completed or failed, as indicated by the OperationStatus parameter above. For a failed Operation, a @Message.ExtendedInfo annotation may be supplied in the response payload. • There is no response payload as indicated by Bit 2 of the OperationExecutionFlags parameter above • The entire payload cannot fit within this message, subject to the maximum transfer chunk size as determined at registration time via the NegotiateMediumParameters command

2937

Type	Response data (continued)
varstring	<p>ETag</p> <p>String data for an ETag digest of the target resource; the string text format shall be UTF-8. The ETag may be skipped (an empty string returned in this field) for any of the following actions: Action, Delete, Replace, and Update. The ETag shall also be skipped (an empty string returned in this field) if execution of the Operation has not yet finished.</p> <p>Additional notes on processing ETags may be found in clause 7.2.3.10.4.</p> <p>NOTE ETags provided via this field are not escaped for inclusion in JSON data as they are primarily intended to be used for the ETag HTML header. MCs should be aware that performing a raw comparison of an ETag retrieved from this command with one received as part of BEJ-encoded JSON data will result in a mismatch as the ETag format requires characters that must be escaped in JSON data.</p>
null or bejEncoding	<p>ResponsePayload</p> <p>The response payload. The format of this parameter shall be null (consisting of zero bytes) if the ResponsePayloadLength above is zero; it shall be bejEncoding otherwise.</p>

2938 **12.6 RDEOperationKill command (0x15) format**

2939 This command enables the MC to request that an RDE Device terminate an Operation. The RDE Device
 2940 shall kill the Operation if the Operation can be killed; however, the MC must be aware that not all
 2941 Operations can be terminated.

2942 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2943 respond with data formatted per the Response Data section if it supports the command.

2944 **Table 69 – RDEOperationKill command format**

Type	Request data
uint32	<p>ResourceID</p> <p>The resourceID of a resource in the Redfish Resource PDR to which the Task’s operation was targeted</p>
rdeOpID	<p>OperationID</p> <p>Identification number for this Operation; must match the one used for all commands relating to this Operation</p>

Type	Request data (continued)
bitfield8	<p>KillFlags</p> <p>Flags for killing the Operation:</p> <p>[7:3] - reserved for future use</p> <p>[2] - discard_results; if 1b and the RDE Device is in the HAVE_RESULTS state for this Operation, the results of the Operation shall be discarded and the Operation state set to Inactive. The MC shall not set the discard_results bit in conjunction with any other bits in the KillFlags. In the event that the MC violates this restriction, the RDE Device shall respond with completion code ERROR_INVALID_DATA and stop processing the request.</p> <p>[1] - run_to_completion; if 1b, the Operation should be run to completion but no further response should be sent to the MC. The MC shall not set the run_to_completion bit without also setting the discard_record bit. In the event that the MC violates this restriction, the RDE Device shall respond with completion code ERROR_INVALID_DATA and stop processing the request.</p> <p>[0] - discard_record; if 1b and the kill command returns success, the RDE Device shall discard internal records associated with this Operation as soon as it is killed; the RDE Device should not expect the MC to call RedfishOperationComplete for this Operation. If the Operation has spawned a Task, the RDE Device shall not create an Event when execution is finished.</p>
Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES, ERROR_OPERATION_ABANDONED, ERROR_OPERATION_FAILED, ERROR_OPERATION_UNKILLABLE, ERROR_NO_SUCH_RESOURCE, ERROR_UNEXPECTED }</p>

2945 **12.7 RDEOperationEnumerate command (0x16) format**

2946 This command enables the MC to request that an RDE Device enumerate all Operations that are
 2947 currently active (not in state INACTIVE in the Operation lifecycle state machine of clause 9.2.3.2). It is
 2948 expected that the MC will typically use this command during its initialization to discover any Operations
 2949 that spawned Tasks that were active through a shutdown.

2950 NOTE When instantiating Operations, the RDE Device shall not create a new Operation if including the total data for
 2951 all Operations would cause the response message for this command to exceed the negotiated maximum transfer
 2952 chunk size (see clause 11.2) for any of the mediums on which the MC has communicated with the RDE Device.

2953 If the RDE Device accepts operations from protocols other than Redfish, it should make them visible as
 2954 RDE Operations while they are active by enumerating them in response to this command.

2955 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2956 respond with data formatted per the Response Data section if it supports the command. For a non-
 2957 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2958

Table 70 – RDEOperationEnumerate command format

Type	Request data
n/a	This request contains no parameters
Type	Response data
enum8	CompletionCode value: { PLDM_BASE_CODES }
uint16	OperationCount The number of active Operations N described in the remainder of this message
uint32	ResourceID [0] The resource ID of the Redfish Resource PDR to which the Operation was targeted. Shall be omitted if OperationCount is zero
rdeOpID	OperationID [0] Operation identifier assigned for the Operation when the MC initialized the Operation via the RDEOperationInit command or when the RDE Device chose to make an external Operation visible via RDE. This field shall be omitted if OperationCount above is zero
enum8	OperationType [0] The type of Operation. Shall be omitted if OperationCount is zero values: { OPERATION_HEAD = 0; OPERATION_READ = 1; OPERATION_CREATE = 2; OPERATION_DELETE = 3; OPERATION_UPDATE = 4; OPERATION_REPLACE = 5; OPERATION_ACTION = 6 } This field shall be omitted if OperationCount above is zero
...	...
uint32	ResourceID [N - 1] The resource ID of the Redfish Resource PDR to which the Operation was targeted
rdeOpID	OperationID [N - 1] Operation identifier assigned for the Operation when the MC initialized the Operation via the RDEOperationInit command or when the RDE Device chose to make an external Operation visible via RDE
enum8	OperationType [N - 1] The type of Operation values: { OPERATION_HEAD = 0; OPERATION_READ = 1; OPERATION_CREATE = 2; OPERATION_DELETE = 3; OPERATION_UPDATE = 4; OPERATION_REPLACE = 5; OPERATION_ACTION = 6 }

2959 **13 PLDM for Redfish Device Enablement – Utility commands**

2960 **13.1 RDEMultipartSend command (0x30) format**

2961 This command enables the MC to send a large volume of data to an RDE Device. In the event of a data
 2962 checksum error, the MC may reissue the first RDEMultipartSend command with the initial data transfer
 2963 handle; the RDE Device shall recognize this to mean that the transfer failed and respond as if this were
 2964 the first transfer attempt. If the MC chooses not to restart the transfer, or in any other error occurs, the MC
 2965 should abandon the transfer. In the latter case, if the transfer is part of an Operation, the MC shall
 2966 explicitly abort and then finalize the Operation via the RDEOperationKill and RDEOperationComplete
 2967 commands (see clauses 12.6 and 12.4).

2968 Similarly, in the event of transient transfer errors for individual chunks of the data, the MC may retry those
 2969 chunks by reissuing the RDEMultiPartSend command corresponding to those chunks provided it has not
 2970 yet issued a RDEMultiPartSend command for a subsequent chunk. When the RDE Device receives a
 2971 request with data formatted per the Request Data section below, it shall respond with data formatted per
 2972 the Response Data section. For a non-SUCCESS CompletionCode with the exception of
 2973 ERROR_BAD_CHECKSUM, only the CompletionCode field of the Response Data shall be returned. In
 2974 the case an ERROR_BAD_CHECKSUM is returned, the RDE Device may set the TransferOperation to
 2975 XFER_FIRST_PART.

2976 NOTE In versions of this specification prior to v1.1.0, this command was named MultiPartSend.

2977 **Table 71 – RDEMultiPartSend command format**

Type	Request data
uint32	<p>DataTransferHandle</p> <p>A handle to uniquely identify the chunk of data to be sent. If TransferFlag below is START or START_AND_END, this must match the SendDataTransferHandle that was supplied by the RDE Device in the response to RDEOperationInit.</p> <p>The DataTransferHandle supplied shall be either the initial handle to begin or restart a transfer or the NextDataTransferHandle as specified in the previous chunk.</p>
rdeOpID	<p>OperationID</p> <p>Identification number for this Operation; must match the one previously used for all commands relating to this Operation; 0x0000 if this transfer is not part of an Operation.</p>
enum8	<p>TransferFlag</p> <p>An indication of current progress within the transfer. The value START_AND_END indicates that the entire transfer consists of a single chunk.</p> <p>value: { START = 0, MIDDLE = 1, END = 2, START_AND_END = 3 }</p>
uint32	<p>NextDataTransferHandle</p> <p>The handle for the next chunk of data for this transfer; zero (0x00000000) if no further data.</p>
uint32	<p>DataLengthBytes</p> <p>The length in bytes N of data being sent in this chunk, including both the Data and DataIntegrityChecksum (if present) fields. This value and the data bytes associated with it shall not cause this request message to exceed the negotiated maximum transfer chunk size (clause 11.2).</p>
uint8	<p>Data [0]</p> <p>The first byte of the current chunk of data. Shall be omitted if only the DataIntegrityChecksum is present.</p>
...	...
uint8	<p>Data [N-1]</p> <p>The last byte of the current chunk of data. Shall be omitted if only the DataIntegrityChecksum is present.</p>

Type	Request data (continued)
uint32	<p>DataIntegrityChecksum</p> <p>32-bit CRC for the entirety of data (all parts concatenated together, excluding this checksum). Shall be omitted for non-final chunks (TransferFlag ≠ END or START_AND_END) in the transfer. The DataIntegrityChecksum shall not be split across multiple chunks. If appending the DataIntegrityChecksum would cause this request message to exceed the negotiated maximum transfer chunk size (clause 11.2), the DataIntegrityChecksum shall be sent as the only data in another chunk.</p> <p>For this specification, the CRC-32 algorithm with the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ (same as the one used by IEEE 802.3) shall be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first.</p>
Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES, ERROR_OPERATION_ABANDONED, ERROR_OPERATION_FAILED, ERROR_UNEXPECTED, ERROR_BAD_CHECKSUM }</p> <p>If the DataTransferHandle does not correspond to a valid chunk, the RDE Device shall return CompletionCode ERROR_INVALID_DATA.</p>
enum8	<p>TransferOperation</p> <p>The follow-up action that the RDE Device is requesting of the MC:</p> <ul style="list-style-type: none"> • XFER_FIRST_PART: resend the initial chunk (restarting the transmission, such as if the checksum of data received did not match the DataIntegrityChecksum in the final chunk) • XFER_NEXT_PART: send the next chunk of data • XFER_ABORT: stop the transmission and do not retry. The MC shall proceed as if the transmission is permanently failed in this case • XFER_COMPLETE: no further follow-up needed, the transmission completed normally <p>value: { XFER_FIRST_PART = 0, XFER_NEXT_PART = 1, XFER_ABORT = 2, XFER_COMPLETE = 3 }</p>

2978 **13.2 RDEMultipartReceive command (0x31) format**

2979 This command enables the MC to receive a large volume of data from an RDE Device. In the event of a
 2980 data checksum error, the MC may reissue the first RDEMultipartReceive command with the initial data
 2981 transfer handle; the RDE Device shall recognize this to mean that the transfer failed and respond as if this
 2982 were the first transfer attempt. If the MC chooses not to restart the transfer, or in any other error occurs,
 2983 the MC should abandon the transfer. In the latter case, if the transfer is part of an Operation, the MC shall
 2984 explicitly abort and finalize the Operation via the RDEOperationKill and then RDEOperationComplete
 2985 commands (see clauses 12.6 and 12.4).

2986 Similarly, in the event of transient transfer errors for individual chunks of the data, the MC may retry those
 2987 chunks by reissuing the RDEMultipartReceive command corresponding to those chunks provided it has
 2988 not yet issued a RDEMultipartReceive command for a subsequent chunk.

2989 When the RDE Device receives a request with data formatted per the Request Data section below, it shall
 2990 respond with data formatted per the Response Data section if it supports the command. For a non-
 2991 SUCCESS CompletionCode, only the CompletionCode field of the Response Data shall be returned.

2992 NOTE In versions of this specification prior to v1.1.0, this command was named MultipartReceive.

2993

Table 72 – RDEMultiPartReceive command format

Type	Request data
uint32	<p>DataTransferHandle</p> <p>A handle to uniquely identify the chunk of data to be retrieved. If TransferOperation below is XFER_FIRST_PART and the OperationID below is zero, this must match the TransferHandle supplied by the RDE Device in the response to the GetSchemaDictionary, GetMessageRegistry, or GetSchemaFile command. If TransferOperation below is XFER_FIRST_PART and the OperationID below is nonzero, this must match the SendDataTransferHandle that was supplied by the RDE Device in the response to RDEOperationInit. If TransferOperation below is XFER_NEXT_PART, this must match the NextDataHandle supplied by the RDE Device with the previous chunk.</p> <p>The DataTransferHandle supplied shall be either the initial handle to begin or restart a transfer or the NextDataTransferHandle supplied with the previous chunk.</p>
rdeOpID	<p>OperationID</p> <p>Identification number for this Operation; must match the one previously used for all commands relating to this Operation; 0x0000 if this transfer is not part of an Operation</p>
enum8	<p>TransferOperation</p> <p>The portion of data requested for the transfer:</p> <ul style="list-style-type: none"> • XFER_FIRST_PART: The MC is asking the transfer to begin or to restart from the beginning • XFER_NEXT_PART: The MC is asking for the next portion of the transfer • XFER_ABORT: The MC is requesting that the transfer be discarded. The RDE Device may discard any internal data structures it is maintaining for the transfer <p>value: { XFER_FIRST_PART = 0, XFER_NEXT_PART = 1, XFER_ABORT = 2 }</p>
Type	Response data
enum8	<p>CompletionCode</p> <p>value: { PLDM_BASE_CODES, ERROR_OPERATION_ABANDONED, ERROR_OPERATION_FAILED, ERROR_UNEXPECTED, ERROR_BAD_CHECKSUM }</p> <p>If the DataTransferHandle does not correspond to a valid chunk, the RDE Device shall return CompletionCode ERROR_INVALID_DATA.</p> <p>If the transfer is aborted, the RDE Device shall acknowledge this status by returning SUCCESS.</p>
enum8	<p>TransferFlag</p> <p>value: { START = 0, MIDDLE = 1, END = 2, START_AND_END = 3 }</p> <p>This field shall be omitted for a non-SUCCESS CompletionCode or if the transfer has been aborted</p>
uint32	<p>NextDataTransferHandle</p> <p>The handle for the next chunk of data for this transfer; zero (0x00000000) if no further data</p> <p>This field shall be omitted for a non-SUCCESS CompletionCode or if the transfer has been aborted</p>
uint32	<p>DataLengthBytes</p> <p>The length in bytes N of data being sent in this chunk, including both the Data and DataIntegrityChecksum (if present) fields. This value and the data bytes associated with it shall not cause this response message to exceed the negotiated maximum transfer chunk size (clause 11.2).</p> <p>This field shall be omitted for a non-SUCCESS CompletionCode or if the transfer has been aborted</p>
uint8	<p>Data [0]</p> <p>The first byte of current chunk of data. Shall be omitted if only the DataIntegrityChecksum is present.</p> <p>This field shall be omitted for a non-SUCCESS CompletionCode or if the transfer has been aborted</p>
...	...

Type	Response data (continued)
uint8	<p>Data [N-1]</p> <p>The last byte of the current chunk of data. Shall be omitted if only the DataIntegrityChecksum is present.</p> <p>This field shall be omitted for a non-SUCCESS CompletionCode or if the transfer has been aborted</p>
uint32	<p>DataIntegrityChecksum</p> <p>32-bit CRC for the entire block of data (all parts concatenated together, excluding this checksum). Shall be omitted for non-final chunks (TransferFlag ≠ END or START_AND_END) in the transfer or for aborted transfers. The DataIntegrityChecksum shall not be split across multiple chunks. If appending the DataIntegrityChecksum would cause this response message to exceed the negotiated maximum transfer chunk size (clause 11.2), the DataIntegrityChecksum shall be sent as the only data in another chunk.</p> <p>For this specification, the CRC-32 algorithm with the polynomial $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$ (same as the one used by IEEE 802.3) shall be used for the integrity checksum computation. The CRC computation involves processing a byte at a time with the least significant bit first.</p>

2994 **14 Additional Information**

2995 **14.1 RDE Multipart transfers**

2996 The various commands defined in clauses 11 and 12 support bulk transfers via the RDEMultipartSend
 2997 and RDEMultipartReceive commands defined in clause 13. The RDEMultipartSend and
 2998 RDEMultipartReceive commands use flags and data transfer handles to perform multipart transfers. A
 2999 data transfer handle uniquely identifies the next part of the transfer. The data transfer handle values are
 3000 implementation specific. For example, an implementation can use memory addresses or sequence
 3001 numbers as data transfer handles.

3002 NOTE If both the RDE Device and the MC support use of PLDM common multipart transfers, those versions of the
 3003 commands shall be used in lieu of the RDE versions. The following notes apply:

- 3004 • All transfers shall consist of a single portion, beginning at offset zero and transferring the entire buffer
- 3005 • The TransferContext field, which is defined in [DSP0240](#) to be protocol specific, shall be supplied with the
 3006 OperationID that would have been used with an RDE version of a multipart transfer
- 3007 • Handling of aborted transfers, which is defined in [DSP0240](#) to be protocol specific, shall follow the notes
 3008 provided within this specification for multipart transfers.

3009 **14.1.1 Flag usage for RDEMultipartSend**

3010 The following list shows some requirements for using TransferOperationFlag, TransferFlag, and
 3011 DataTransferHandle in RDEMultipartSend data transfers:

- 3012 • To prepare a large data send for use in an RDE command, a DataTransferHandle shall be sent
 3013 by the MC in the request message of the RDEOperationInit command.
- 3014 • To reflect a data transfer (re)initiated with a RDEMultipartSend command, the
 3015 TransferOperation shall be set to XFER_FIRST_PART in the response message.
- 3016 • For transferring a part after the first part of data, the TransferOperation shall be set to
 3017 XFER_NEXT_PART and the DataTransferHandle shall be set to the NextDataTransferHandle
 3018 that was obtained in the request for the previous RDEMultipartSend command for this data
 3019 transfer.
- 3020 • The TransferFlag specified in the request for a RDEMultipartSend command has the following
 3021 meanings:

- 3022 – START, which is the first part of the data transfer
- 3023 – MIDDLE, which is neither the first nor the last part of the data transfer
- 3024 – END, which is the last part of the data transfer
- 3025 – START_AND_END, which is the first and the last part of the data transfer. In this case, the
3026 transfer consists of a single chunk
- 3027 • For a RDEMultipartSend, the requester shall consider a data transfer complete when it receives
3028 a success CompletionCode in the response to a request in which the TransferFlag was set to
3029 End or StartAndEnd.

3030 14.1.2 Flag usage for RDEMultipartReceive

3031 The following list shows some requirements for using TransferOperationFlag, TransferFlag, and
3032 DataTransferHandle in RDEMultipartReceive data transfers:

- 3033 • To prepare a large data transfer receive for use in an RDE command, a DataTransferHandle
3034 shall be sent by the RDE Device in the response message to the RDEOperationInit,
3035 SupplyCustomRequestParameters, or RDEOperationStatus command after an Operation has
3036 finished execution and results are ready for pick-up.
- 3037 • To initiate a data transfer with a RDEMultipartReceive command, the TransferOperation shall
3038 be set to XFER_FIRST_PART in the request message.
- 3039 • For transferring a part after the first part of data, the TransferOperation shall be set to
3040 XFER_NEXT_PART and the DataTransferHandle shall be set to the NextDataTransferHandle
3041 that was obtained in the response to the previous RDEMultipartReceive command for this data
3042 transfer.
- 3043 • The TransferFlag specified in the response of a RDEMultipartReceive command has the
3044 following meanings:
 - 3045 – START, which is the first part of the data transfer
 - 3046 – MIDDLE, which is neither the first nor the last part of the data transfer
 - 3047 – END, which is the last part of the data transfer
 - 3048 – START_AND_END, which is the first and the last part of the data transfer
- 3049 • For a RDEMultipartReceive, the requester and responder shall consider a data transfer
3050 complete when the TransferFlag in the response is set to END or START_AND_END. After this
3051 point, the transfer may not be restarted without repeating the invoking commands, such as
3052 GetSchemaDictionary for a multipart transfer of a dictionary.

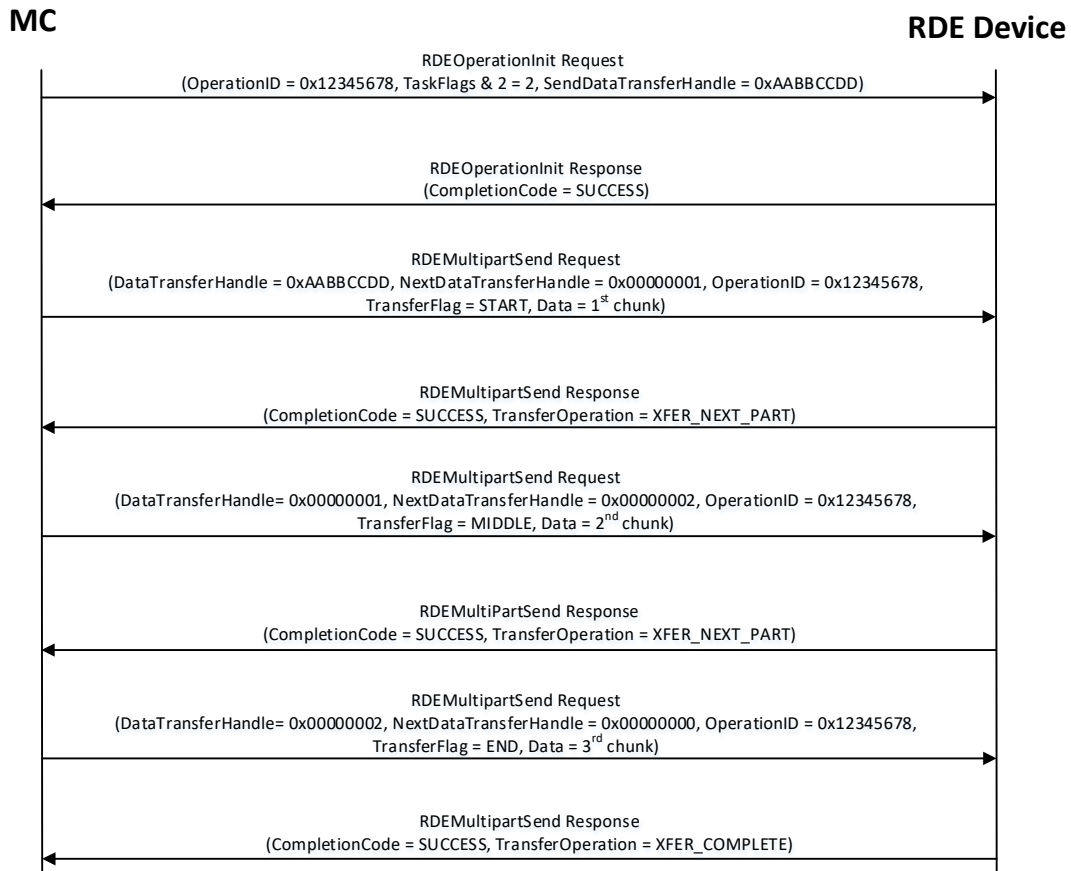
3053 14.1.3 RDE Multipart transfer examples

3054 The following examples show how the multipart transfers can be performed using the generic mechanism
3055 defined in the commands.

3056 In the first example, the MC sends data to the RDE Device as part of a Redfish Update operation.
3057 Following the RDEOperationInit command sequence, the MC effects the transfer via a series of
3058 RDEMultipartSend commands. Figure 18 shows the flow of the data transfer.

3059 In the second example, the MC retrieves the dictionary for a schema. The request is initiated via the
3060 GetSchemaDictionary command and then effected via one or more RDEMultipartReceive commands.

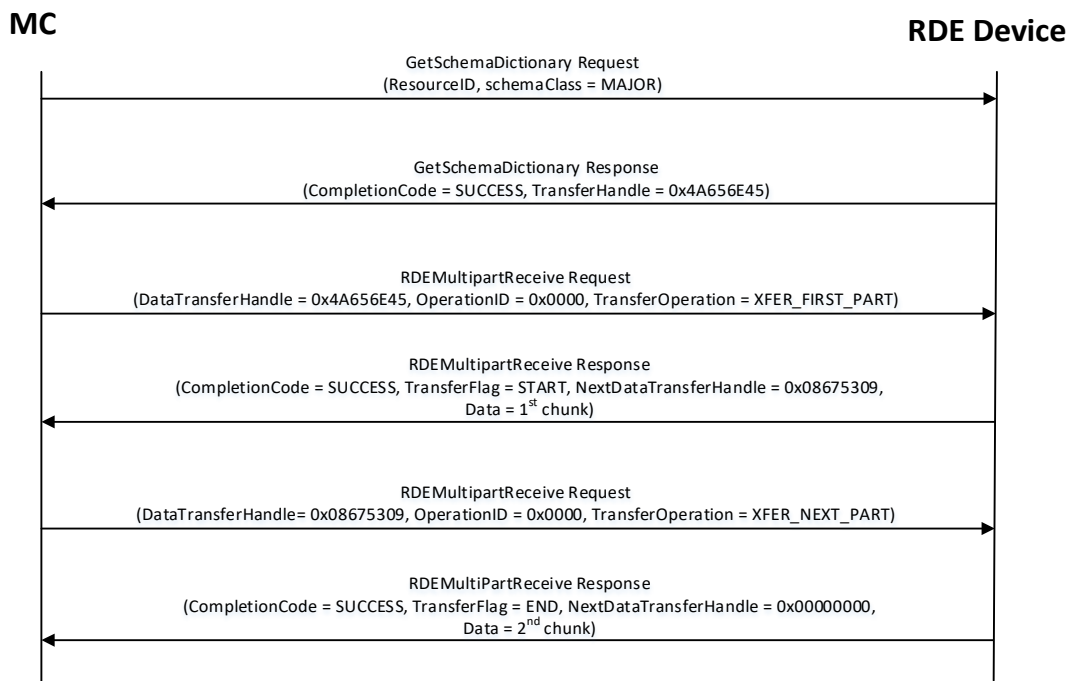
3061 Figure 19 shows the flow of the data transfer.



3062

3063

Figure 18 – RDEMultiPartSend example



3064

3065 **Figure 19 – RDEMultipartReceive example**

3066 **14.2 Implementation notes**

3067 Several implementation notes apply to manufacturers of RDE Devices or of management controllers.

3068 **14.2.1 Schema updates**

3069 If one or more schemas for an RDE Device are updated, the RDE Device may communicate this to the
3070 MC by triggering an event for the affected PDRs. When the MC detects a PDR update, it shall reread the
3071 affected PDRs.

3072 **14.2.2 Storage of dictionaries**

3073 It is not necessary for the MC to maintain all dictionaries in memory at any given time. It may flush
3074 dictionaries at will since they can be retrieved on demand from the RDE Devices via the
3075 GetSchemaDictionary command (clause 11.2). However, if the MC has to retrieve a dictionary “on
3076 demand” to support a Redfish query, this will likely incur a performance delay in responding to the client.
3077 For MCs with highly limited memory that cannot retain all the dictionaries they need to support, care must
3078 thus be exercised in the runtime selection of dictionaries to evict. Such caching considerations are
3079 outside the scope of this specification.

3080 **14.2.3 Dictionaries for related schemas**

3081 MCs must not assume that sibling instances of Redfish Resource PDRs in a hierarchy (such as collection
3082 members) use the same version of a schema. They could, for example, correspond to individual elements
3083 from an array of hardware (such as a disk array) built by separate manufacturers and supporting different
3084 versions of a major schema or with different OEM extensions to it. However, at such time as the MC has
3085 verified that two siblings do in fact use the same schemas, there is no reason to store multiple copies of
3086 the dictionary corresponding to that schema. Of course, sibling instances of resources stored within the
3087 same PDR share all dictionaries; it is only with instances of resources from separate PDRs that this
3088 applies.

3089 Similarly, it is expected to be fairly commonplace that the system managed by an MC could have multiple
3090 RDE Devices of the same class, such as multiple network adapters or multiple RAID array controllers. In
3091 such cases, however, there is no guarantee that each such RDE Device will support the same version of
3092 any given Redfish schema.

3093 To handle such cases, MCs have two choices. The most straightforward approach is to simply maintain
3094 each dictionary associated with the RDE Device it came from. This of course has space implications. A
3095 more practical approach is to store one copy of the dictionary for each version of the schema and then
3096 keep track of which version of the dictionary to use with which RDE Device. Because RDE Devices may
3097 support only subsets of the properties in resources, care must be taken when employing this approach to
3098 ensure that all supported properties are covered in the dictionaries selected. This may be done by
3099 merging dictionaries at runtime, though details of how to merge dictionaries are out of scope for this
3100 specification. In particular, OEM sections of dictionaries are not generally able to be merged as the
3101 sequence numbers for the names of the different OEM extensions themselves are likely to overlap.

3102 However, an even better approach is available. In Redfish schemas, so long as only the minor and
3103 release version numbers change, schemas are required to be fully backward compatible with earlier
3104 revisions. Individual properties and enumeration values may be added but never removed. The MC can
3105 therefore leverage this to retain only the newest instance of dictionary for each major version supported
3106 by RDE Devices. Again, the fact that RDE Devices may support only subsets of the properties in a
3107 resource means that care must be taken to ensure dictionary support for all the properties used across all
3108 RDE Devices that implement any given schema.

3109 14.2.4 [MC] HTTP/HTTPS POST Operations

3110 As specified in [DSP0266](#), a Redfish POST Operation can represent either a Create Operation or an
3111 Action. To distinguish between these cases, the MC may examine the URI target supplied with the
3112 operation. If it points to a collection, the MC may assume that the Operation is a Create; if it points to an
3113 action, the MC may assume the Operation is an Action. Alternatively, the MC may presuppose that the
3114 POST is a Create Operation and if it receives an ERROR_WRONG_LOCATION_TYPE error code from
3115 the RDE Device, retry the Operation as an Action. This second approach reduces the amount of URI
3116 inspection the MC has to perform in order to proxy the Operation at the cost of a small delay in
3117 completion time for the Action case. (The supposition that POSTs correspond to Create Operations could
3118 of course be reversed, but it is expected that Actions will be much rarer than Create Operations.)
3119 Implementers should be aware that such delays could cause a client-side timeout.

3120 Another clue that could be used to differentiate between POSTs intended as create operations vs POSTs
3121 intended as actions would be trial encodings of supplied payload data. If there is no payload data, then
3122 the request is either in error or an action. In this case, the payload should be encoded with the dictionary
3123 for the major schema associated with target resource. On the other hand, if the payload is intended for a
3124 create operation, the correct dictionary to use would be the collection member dictionary, which may be
3125 retrieved via the GetSchemaDictionary command (clause 11.2), specifying
3126 COLLECTION_MEMBER_TYPE as the dictionary to retrieve.

3127 14.2.4.1 Support for Actions

3128 When a Redfish client issues a Redfish Operation for an Action, the URI target for the Action will be a
3129 POST of the form /redfish/v1/{path to root of RDE Device component}/{path to RDE Device owned
3130 resource}/Actions/schema_name.action_name. To process this, the MC may translate {path to root of
3131 RDE Device component} and {path to RDE Device owned resource} normally to identify the PDR against
3132 which the Operation should be executed. (If the URI is not in this format, this is another indication that the
3133 POST operation is probably a CREATE.) After it has performed this step, the MC can then check its PDR
3134 hierarchy to find the Redfish Action PDR containing an action named schema_name.action_name. If it
3135 doesn't find one, the MC shall respond with HTTP status code 404, Not Found and stop processing the
3136 Operation.

3137 After the correct Action is located, the MC can translate any request parameters supplied with the Action.
3138 To do so, it should look within the dictionary at the point beginning with the named action, and then
3139 navigate into the Parameters set under the action. From there, standard encoding rules apply. When
3140 supplying a locator for the Action to the RDE Device as part of the RDEOperationInit command, the MC
3141 shall not include the Parameters set as one of the sequence numbers comprising the locator; rather, it
3142 shall stop with the sequence number for the property corresponding to the Action's name.

3143 After the Action is complete, it may contain result parameters. If present, definitions for these will be found
3144 in the dictionary in a ReturnType set parallel to the Parameters set that contained any request
3145 parameters. If an Action does not contain explicit result parameters, the ReturnType set will generally not
3146 be present in the dictionary. The structure of the ReturnType set mirrors exactly that of the Parameters
3147 set.

3148 14.2.5 Consistency checking of read Operations

3149 Because the collection of data contained within a schema cannot generally be read atomically by RDE
3150 Devices, issues of consistency arise. In particular, if the RDE Device reads some of the data, performs an
3151 update, and then reads more data, there is no guarantee that data read in the separate "chunks" will be
3152 mutually consistent. While the level of risk that this could pose for a client consumer of the data may vary,
3153 the threat will not. The problem is exacerbated when reads must be performed across multiple resources
3154 in order to satisfy a client request: The window of opportunity for a write to slip in between distinct
3155 resource reads is much larger than the window between reads of individual pieces of data in a single
3156 resource.

3157 To resolve the threat of inconsistency, MCs should utilize a technique known as consistency checking.
3158 Before issuing a read, the MC should retrieve the ETag for the schema to be read, using the
3159 GetResourceETag command (clause 11.5). For a read that spans multiple resources, the global ETag
3160 should be read instead, by supplying 0xFFFFFFFF for the ResourceID in the command. The MC should
3161 then proceed with all of the reads and then check the ETag again. If the ETag matches what was initially
3162 read, the MC may conclude that the read was consistent and return it to the client. Otherwise, the MC
3163 should retry. It is expected that consistency failures will be very rare; however, if after three attempts, the
3164 MC cannot obtain a consistent read, it should report error 500, Internal Server Error to the client.

3165 NOTE For reads that only span a single resource, if the RDE Device asserts the **atomic_resource_read** bit in the
3166 **DeviceCapabilitiesFlags** response message to the NegotiateRedfishParameters command (clause 11.1), the MC
3167 may skip consistency checking.

3168 **14.2.6 [MC] Placement of RDE Device resources in the outward-facing Redfish** 3169 **URI hierarchy**

3170 In the Redfish Resource PDRs and Redfish Entity Association PDRs that an RDE Device presents, there
3171 will normally be one or a limited number that reflect EXTERNAL (0x0000) as their ContainingResourceID.
3172 These resources need to be integrated into the outward-facing Redfish URI hierarchy. Resources that do
3173 not reflect EXTERNAL as their ContainingResourceID do not need to be placed by the MC; it is the RDE
3174 Device's responsibility to make sure that they are accessible via some chain of Redfish Resource and
3175 Redfish Entity Association PDRs (including PDRs chained via @link properties) that ultimately link to
3176 EXTERNAL.

3177 When retrieving these PDRs for RDE Device components, the MC should read the
3178 ProposedContainingResourceName from the PDR. While following this recommendation is not
3179 mandatory, the MC should use it to inform a placement decision. If the MC does not follow the placement
3180 recommendation, it should read the MajorSchemaName field to identify the type of RDE Device they
3181 correspond to. Within the canon of standard Redfish schemas, there are comparatively few that reside at
3182 the top level, and each has a well-defined place it should appear within the hierarchy. The MC should
3183 thus make a simple map of which top-level schema types map to which places in the hierarchy and use
3184 this to place RDE Devices. In making these placement decisions, the MC should take information about
3185 the hardware platform topology into account so as to best reflect the overall Redfish system.

3186 It may happen that the MC encounters a schema it does not recognize. This can occur, for example, if a
3187 new schema type is standardized after the MC firmware is built. The handling of such cases is up to the
3188 MC. One possibility would be to place the schema in the OEM section under the most appropriate
3189 subobject. For an unknown DMTF standard schema, this should be the OEM/DMTF object. (To tell that a
3190 schema is DMTF standard, the MC may retrieve the published URI via GetSchemaURI command of
3191 clause 11.4, download the schema, and inspect the schema, namespace, or other content.)

3192 Naturally, wherever the MC places the RDE Device component, it shall add a link to the RDE Device
3193 component in the JSON retrieved by a client from the enclosing location.

3194 **14.2.7 LogEntry and LogEntryCollection resources**

3195 RDE Devices that support the LogEntry and LogEntryCollection resources must be aware that large
3196 volumes of LogEntries can overwhelm the 16 bit ResourceID space available for identifying Redfish
3197 Resource PDRs. To handle this case, it is recommended that RDE Devices provide a PDR for the
3198 LogEntryCollection but do NOT provide PDRs for the individual LogEntry instances. Instead, RDE
3199 Devices that support these schemas should also support the link expansion query parameter (see \$levels
3200 in [DSP0266](#) and the LinkExpand parameter from SupplyCustomRequestParameters in clause 12.2). This
3201 means that they should fill out the related resource links in the "Members" section of the response with
3202 bejResourceLinkExpansion data in which the encoded ResourceID is set to zero to ensure that the MC
3203 gets the COLLECTION_MEMBER_TYPE dictionary from the LogEntryCollection.

3204 **14.2.8 On-demand pagination**

3205 In Redfish, certain read operations may produce a very large amount of data. For example, reading a
3206 collection with many members will produce output with size proportional to the number of members.
3207 Rather than overload clients with a huge transfer of data, Redfish Devices may paginate it into chunks
3208 and provide one page at a time with an @odata.nextlink annotation giving a URI from which to retrieve
3209 the next piece.

3210 RDE supports the same pagination approach. It is entirely at the RDE Device's discretion whether to
3211 paginate and where to draw pagination boundaries. When the RDE Device wishes to paginate, it shall
3212 insert an @odata.nextlink annotation, using a deferred binding pagination reference (see
3213 \$LINK.PDR<resource-ID>.PAGE<pagination-offset>% in clause 8.3), filling in the next page number for
3214 the data being returned. When the MC decodes this deferred binding, it shall create a temporary URI for
3215 the pagination and expose this pagination URI in the decoded JSON response it sends back to the client.
3216 Naturally, the encoded pagination URI must be decodable to extract the page number. Finally, when the
3217 client attempts a read from the pagination URI, the MC shall extract out the page number and send it to
3218 the RDE Device via the PaginationOffset field in the request message for the
3219 SupplyCustomRequestParameters command (clause 12.2).

3220 **14.2.9 Considerations for Redfish clients**

3221 No changes to behavior are required of Redfish clients in order to interact with BEJ-based RDE Devices;
3222 the details of providing them to the client are completely transparent from the client perspective. In fact, a
3223 fundamental design goal of this specification is that it should be impossible for a client to tell whether a
3224 Redfish message was ultimately serviced by an RDE Device that operates in JSON over HTTP/HTTPS or
3225 BEJ over PLDM.

3226

3227 **14.2.10 OriginOfCondition in Redfish events**

3228 The OriginOfCondition field in the Redfish event schema contains a link reference to a Redfish resource
3229 associated with a Redfish event. In typical use cases, resource data is read upon receiving the event to
3230 determine the resource state when the event transpired. This can happen either explicitly, from the client
3231 performing a read on the OriginOfCondition resource, or implicitly, if IncludeOriginOfCondition is set in the
3232 EventDestination when the client registered for Redfish events.

3233 RDE version 1.1 does not provide support for a device to populate the OriginOfCondition field with full
3234 resource data. However, an MC that wishes to minimize the timing window for the race condition may
3235 perform the appropriate read immediately upon receiving the Redfish event.

3236 **14.2.11 [MC] Merging dictionaries with OEM extensions**

3237 When merging dictionaries, MCs should consider that OEM extensions to Redfish schemas are
3238 enumerated alphabetically. In particular, the root objects (sets) of extensions (which come immediately
3239 under "OEM" inside the root object of the host schema) are likely to have conflicting sequence numbers if
3240 different sets of extensions appear in two different dictionaries for a given host schema.

3241 Additionally, no attempt has been made in this specification to make registry dictionaries able to be
3242 merged.

3243 **14.2.12 PATCH on Array Properties**

3244 The Redfish standard ([DSP0266](#)) defines special behavior for Redfish PATCH Operations when applied
3245 to Array properties. All RDE Devices that support PATCHable Resources containing writable arrays shall
3246 support special encodings as described in this clause.

3247 NOTE The special support described in this clause was not detailed in specification versions prior to version 1.2.0.
3248 As a result, there is no guarantee that RDE devices that implement older versions of the specification will
3249 understand requests formatted in this fashion. In the event that a requested PATCH Operation requires
3250 special handling as detailed in this clause and the target device does not support at least specification
3251 version 1.2.0, the MC shall either: reject the PATCH Operation request as unsupported; or fetch the relevant
3252 array, manually build the updated form corresponding to the request, and then submit to the RDE Device a
3253 modified PATCH Operation containing the manually constructed array data.

3254 Within a PATCH request, the service shall accept null to remove an element. This null shall be encoded
3255 by the MC as a bejNULL field even if the dictionary entry for the array does not permit null data. The RDE
3256 Device shall accept a bejNULL in this context for the purpose of deleting an array element even if the
3257 dictionary does not permit null data. Array properties that use the fixed or variable length style, as defined
3258 [DSP0266](#), shall remove those elements, while array properties that use the rigid style, as defined
3259 [DSP0266](#), shall replace removed elements with null elements.

3260 Similarly, within a PATCH request, the service shall accept an empty object {} to leave an element
3261 unchanged. The RDE Device shall accept and interpret an empty object for this purpose.

3262 As defined [DSP0266](#), a Redfish PATCH Operation may indicate the maximum size of an array by
3263 padding null elements at the end of the array sequence. As with deletions mentioned earlier in this
3264 clause, bejNULL encodings shall be used for this purpose by the MC and accepted by the RDE Device
3265 even if the dictionary for the corresponding resource does not permit NULL data.

3266 When processing a PATCH request, an RDE Device shall perform array modification operations in the
3267 following order:

- 3268 1. Modifications
- 3269 2. Deletions
- 3270 3. Additions

3271 As defined [DSP0266](#), a Redfish PATCH Operation with fewer elements than in the current array shall
3272 remove the remaining elements of the array. The RDE Device shall comply to this requirement.

**ANNEX A
(normative)**

Change log

3273
3274
3275
3276

3277

Version	Date	Description
1.0.0	2019-06-25	Released as DMTF Standard
1.0.1	2019-12-09	Errata update
1.1.0	2020-11-19	<ul style="list-style-type: none"> • Added support for nested annotations • Enhanced message registry support, including a new registry dictionary, BEJ encoding • Improved ability to identify OEM extensions to schemas • Added support for PLDM common multipart transfers
1.1.1	2021-10-27	Errata update
1.1.2	2022-10-26	Errata update
1.2.0	2024-06-21	<ul style="list-style-type: none"> • Added support for Redfish Parallel Resource PDR • Clarified support for Redfish array element manipulation via null and \emptyset • Errata updates

3278