

12 Copyright Notice

13 Copyright © 2009-2011, 2016, 2019 DMTF. All rights reserved.

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

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28 withdrawn or modified after publication, and shall be indemnified and held harmless by any party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

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

34 PCI-SIG, PCIe, and the PCI HOT PLUG design mark are registered trademarks or service marks of PCI-
35 SIG.

36 All other marks and brands are the property of their respective owners.

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

CONTENTS

39 Foreword 10

40 Introduction..... 11

41 1 Scope 13

42 2 Normative references 13

43 3 Terms and definitions 14

44 4 Symbols and abbreviated terms..... 16

45 5 Conventions 16

46 6 PLDM for Platform Monitoring and Control version 17

47 7 PLDM for Platform Monitoring and Control overview 17

48 8 PDR architecture 19

49 8.1 General 19

50 8.2 Primary PDR Repository and Device PDR repositories 19

51 8.3 Use of PDRs 19

52 9 Entities..... 23

53 9.1 Entity Identification Information..... 23

54 9.2 Entity Type and Entity IDs 24

55 9.3 Entity Instance Numbers..... 25

56 9.4 Container ID..... 25

57 9.5 Use of Container ID in PDRs 25

58 10 PLDM associations..... 26

59 10.1 Association examples 26

60 10.2 Internal and External Associations 26

61 10.3 Sensor/Effecter to Entity associations 27

62 10.4 FRU Record Set to Entity associations..... 29

63 11 Entity Association PDRs..... 30

64 11.1 Physical-to-Physical containment associations 30

65 11.2 Entity identification relationships between PDRs 32

66 11.3 Linked Entity Association PDRs 33

67 11.4 Logical containment associations 34

68 11.5 Sensor/effecter associations with logical entities 35

69 11.6 Merged entity associations 36

70 11.7 Separation of logical and physical associations 38

71 11.8 Designing association PDRs for monitoring and control 38

72 11.9 Terminus associations 39

73 11.10 Interrupt associations..... 42

74 12 PLDM terminus..... 43

75 12.1 TIDs, PLDM Terminus Handles, and Terminus Locator PDRs 44

76 12.2 Requirements for unique TIDs..... 44

77 12.3 Terminus messaging requirements 44

78 12.4 Terminus Locator PDRs..... 44

79 12.5 Enumerating termini..... 45

80 13 PLDM events..... 46

81 13.1 PLDM Event Messages 47

82 13.2 PLDM Event Receiver..... 47

83 13.3 PLDM Event Logging..... 48

84 13.4 PLDM Event Log clearing policies 48

85 13.5 Oldest and newest log entries 49

86 13.6 Event Receiver Location 49

87 13.7 PLDM Event Log entry formats..... 49

88 13.8 PLDM Platform Event Entry Data format 50

89	13.9 OEM Timestamped Event Entry Data format	51
90	13.10 OEM Event Entry Data format	51
91	14 Discovery Agent	51
92	14.1 Assignment of TIDs and Event Receiver location	52
93	14.2 UUIDs for devices in hot-plug or add-in card applications.....	53
94	14.3 UID implementation	53
95	14.4 More than one terminus in a device.....	53
96	14.5 Examples of PDR and UUID use with add-in cards	53
97	15 Initialization Agent	56
98	15.1 General	56
99	15.2 PLDM and power state interaction.....	56
100	15.3 RunInitAgent command	56
101	15.4 Recommended Initialization Agent steps.....	57
102	16 Terminus and event commands.....	57
103	16.1 SetTID command.....	58
104	16.2 GetTID command.....	59
105	16.3 GetTerminusUID command	59
106	16.4 SetEventReceiver command	60
107	16.5 GetEventReceiver command.....	62
108	16.6 PlatformEventMessage command.....	63
109	16.7 PollForPlatformEventMessage command	64
110	16.8 EventMessageSupported Command.....	67
111	16.9 EventMessageBufferSize Command.....	68
112	16.10 eventData format for sensorEvent	69
113	16.11 eventData format for effecterEvent.....	70
114	16.12 eventData format for redfishTaskExecutedEvent	71
115	16.13 eventData format for redfishMessageEvent	71
116	16.14 eventData format for pldmPDRRepositoryChgEvent	72
117	16.15 eventData format for pldmMessagePollEvent	74
118	16.16 eventData format for heartbeatTimerElapsedEvent	75
119	17 PLDM Numeric Sensors.....	75
120	17.1 Sensor readings, data sizes	75
121	17.2 Units and reading conversion	75
122	17.3 Reading-only or threshold-based numeric sensors	76
123	17.4 Readable and settable thresholds	76
124	17.5 Update/polling intervals and states updates.....	76
125	17.6 Thresholds, Present State, and Event State	76
126	17.7 Manual re-arm and auto re-arm sensors	78
127	17.8 Event message generation	78
128	17.9 Threshold values and hysteresis	78
129	18 PLDM Numeric Sensor commands.....	80
130	18.1 SetNumericSensorEnable command.....	80
131	18.2 GetSensorReading command	81
132	18.3 GetSensorThresholds command.....	84
133	18.4 SetSensorThresholds command	85
134	18.5 RestoreSensorThresholds command	86
135	18.6 GetSensorHysteresis command.....	86
136	18.7 SetSensorHysteresis command	87
137	18.8 InitNumericSensor command	88
138	19 PLDM State Sensors.....	89
139	20 PLDM State Sensor commands.....	90
140	20.1 SetStateSensorEnables command.....	90
141	20.2 GetStateSensorReadings command	91
142	20.3 InitStateSensor command	93

143	21	PLDM effecters.....	94
144	21.1	PLDM State Effecters	94
145	21.2	PLDM Numeric Effecters	95
146	21.3	Effector semantics	95
147	21.4	PLDM and OEM effector semantic IDs.....	96
148	22	PLDM effector commands.....	96
149	22.1	SetNumericEffectorEnable command.....	97
150	22.2	SetNumericEffectorValue command.....	97
151	22.3	GetNumericEffectorValue command	98
152	22.4	SetStateEffectorEnables command.....	99
153	22.5	SetStateEffectorStates command.....	101
154	22.6	GetStateEffectorStates command	102
155	23	PLDM Event Log commands	103
156	23.1	GetPLDMEventLogInfo command	104
157	23.2	EnablePLDMEventLogging command.....	106
158	23.3	ClearPLDMEventLog command	106
159	23.4	GetPLDMEventLogTimestamp command	107
160	23.5	SetPLDMEventLogTimestamp command.....	108
161	23.6	ReadPLDMEventLog command	109
162	23.7	GetPLDMEventLogPolicyInfo command	111
163	23.8	SetPLDMEventLogPolicy command.....	114
164	23.9	FindPLDMEventLogEntry command	115
165	24	PLDM State Sets.....	117
166	25	Platform Descriptor Records (PDRs)	117
167	25.1	PDR Repository updates	117
168	25.2	Internal storage and organization of PDRs.....	118
169	25.3	PDR types.....	118
170	25.4	PDR record handles.....	118
171	25.5	Accessing PDRs	118
172	26	PDR Repository commands.....	118
173	26.1	GetPDRRepositoryInfo command	119
174	26.2	GetPDR command.....	121
175	26.3	FindPDR command.....	124
176	26.4	RunInitAgent command	130
177	26.5	GetPDRRepositorySignature command.....	130
178	27	PDR definitions.....	131
179	27.1	Sensor types	131
180	27.2	Effector types	131
181	27.3	State sets	131
182	27.4	Sensor and effector units.....	132
183	27.5	Counters	135
184	27.6	Accuracy, tolerance, resolution, and offset.....	135
185	27.7	Numeric reading conversion formula	141
186	27.8	Numeric effector conversion formula	142
187	28	Platform Descriptor Record (PDR) formats.....	142
188	28.1	Common PDR header format	142
189	28.2	PDR type values	143
190	28.3	Terminus Locator PDR	144
191	28.4	Numeric Sensor PDR.....	147
192	28.5	Numeric Sensor Initialization PDR.....	153
193	28.6	State Sensor PDR.....	154
194	28.7	State Sensor Initialization PDR.....	156
195	28.8	Sensor Auxiliary Names PDR	159
196	28.9	OEM Unit PDR.....	160
197	28.10	OEM State Set PDR	161

198 28.11 Numeric Effector PDR..... 163
 199 28.12 Numeric Effector Initialization PDR..... 168
 200 28.13 State Effector PDR..... 169
 201 28.14 State Effector Initialization PDR..... 170
 202 28.15 Effector Auxiliary Names PDR 173
 203 28.16 OEM Effector Semantic PDR..... 174
 204 28.17 Entity Association PDR 175
 205 28.18 Entity Auxiliary Names PDR 176
 206 28.19 OEM EntityID PDR..... 177
 207 28.20 Interrupt Association PDR 178
 208 28.21 Event Log PDR 179
 209 28.22 FRU Record Set PDR 180
 210 28.23 OEM Device PDR 181
 211 28.24 OEM PDR 182
 212 28.25 Compact Numeric Sensor PDR 183
 213 28.26 Redfish Resource PDR..... 185
 214 28.27 Redfish Entity Association PDR..... 188
 215 28.28 Redfish Action PDR 189
 216 29 Timing..... 190
 217 30 PLDM Command numbers..... 190
 218 ANNEX A (informative) Change log 192
 219 Bibliography 193
 220

221 **Figures**

222 Figure 1 – PLDM used for access only 20
 223 Figure 2 – PLDM with device PDRs..... 21
 224 Figure 3 – PLDM with PDRs for subsystem..... 22
 225 Figure 4 – Entity Identification Information..... 23
 226 Figure 5 – Entity Identification Information format 23
 227 Figure 6 – Entity Identification Information in a Numeric Sensor PDR 27
 228 Figure 7 – Entity Identification Information in a FRU Record Set PDR..... 29
 229 Figure 8 – Physical containment entity association PDR 31
 230 Figure 9 – containerID relationships 32
 231 Figure 10 – Entity identification relationship between PDRs 33
 232 Figure 11 – Linked Entity Association PDRs 34
 233 Figure 12 – Logical Containment PDR 35
 234 Figure 13 – Sensor/effector to logical entity association 36
 235 Figure 14 – Merged entity association PDR example..... 37
 236 Figure 15 – Block diagram for merged entity association PDR example 38
 237 Figure 16 – TID and PLDM Terminus Handle associations..... 40
 238 Figure 17 – Block diagram of Terminus-to-Sensor associations 41
 239 Figure 18 – Received interrupt association example..... 43
 240 Figure 19 – Example of TID and PLDM Terminus Handle relationships 45
 241 Figure 20 – Hot-plug add-in card with single PLDM terminus 54
 242 Figure 21 – Hot-plug add-in card with multiple PLDM termini 55
 243 Figure 22 – Numeric sensor threshold and hysteresis relationships 79
 244 Figure 23 – Accuracy, tolerance, and resolution example..... 136
 245 Figure 24 – Figuring resolution from the design 139

246

247 **Tables**

248	Table 1 – PLDM monitoring and control data types.....	16
249	Table 2 – Parts of the Entity Identification Information format.....	24
250	Table 3 – Field & value descriptions for Entity Identification Information in a Numeric Sensor PDR.....	28
251	Table 4 – Field and value descriptions for Entity Identification Information in a FRU Record Set PDR.....	29
252	Table 5 – PLDM Event Log clearing policies.....	48
253	Table 6 – PLDM Event Log entry format.....	50
254	Table 7 – Platform Event Entry Data format.....	50
255	Table 8 – OEM Timestamped Event Entry Data format.....	51
256	Table 9 – OEM Event Entry Data format.....	51
257	Table 10 – Terminus and event commands.....	57
258	Table 11 – PLDM Event Types.....	58
259	Table 12 – GetTerminusUID command format.....	59
260	Table 13 – SetEventReceiver command format.....	60
261	Table 14 – GetEventReceiver command format.....	62
262	Table 15 – PlatformEventMessage command format.....	63
263	Table 16 – PollForPlatformEventMessage command format.....	66
264	Table 17 – EventMessageSupported command format.....	67
265	Table 18 – EventMessageBufferSize command format.....	68
266	Table 19 – sensorEvent class eventData format.....	69
267	Table 20 – effectorEvent class eventData format.....	71
268	Table 22 – redfishMessageEvent class eventData format.....	72
269	Table 23 – pldmPDRRepositoryChgEvent class eventData format.....	73
270	Table 24 – pldmPDRRepositoryChgEvent changeRecord format.....	74
271	Table 25 – pldmMessagePollEvent class eventData format.....	74
272	Table 26 – heartbeatTimerElapsedEvent class eventData format.....	75
273	Table 27 – Threshold severity levels.....	77
274	Table 28 – Numeric Sensor commands.....	80
275	Table 29 – SetNumericSensorEnable command format.....	80
276	Table 30 – GetSensorReading command format.....	81
277	Table 31 – GetSensorThresholds command format.....	84
278	Table 32 – SetSensorThresholds command format.....	85
279	Table 33 – RestoreSensorThresholds command format.....	86
280	Table 34 – GetSensorHysteresis command format.....	87
281	Table 35 – SetSensorHysteresis command format.....	88
282	Table 36 – InitNumericSensor command format.....	89
283	Table 37 – State Sensor commands.....	90
284	Table 38 – SetStateSensorEnables command format.....	90
285	Table 39 – SetStateSensorEnables opField format.....	91
286	Table 40 – GetStateSensorReadings command format.....	92
287	Table 41 – GetStateSensorReadings stateField format.....	92
288	Table 42 – InitStateSensor command format.....	93
289	Table 43 – InitStateSensor initField format.....	94
290	Table 44 – Categories for effector semantics.....	95

291 Table 45 – State and Numeric Effector commands 96

292 Table 46 – SetNumericEffectorEnable command format 97

293 Table 47 – SetNumericEffectorValue command format 97

294 Table 48 – GetNumericEffectorValue command format 98

295 Table 49 – SetStateEffectorEnables command format..... 100

296 Table 50 – SetStateEffectorEnables opField format..... 100

297 Table 51 – SetStateEffectorStates command format 101

298 Table 52 – SetStateEffectorStates stateField format..... 101

299 Table 53 – GetStateEffectorStates command format 102

300 Table 54 – GetStateEffectorStates stateField format 102

301 Table 55 – PLDM Event Log commands 103

302 Table 56 – GetPLDMEventLogInfo command format..... 104

303 Table 57 – EnablePLDMEventLogging command format..... 106

304 Table 58 – ClearPLDMEventLog command format 107

305 Table 59 – GetPLDMEventLogTimestamp command format 107

306 Table 60 – SetPLDMEventLogTimestamp command format 108

307 Table 61 – ReadPLDMEventLog command format 110

308 Table 62 – PLDMEventLogData format 111

309 Table 63 – GetPLDMEventLogPolicyInfo command format 112

310 Table 64 – SetPLDMEventLogPolicy command format..... 114

311 Table 65 – FindPLDMEventLogEntry command format 116

312 Table 66 – PDR Repository commands..... 118

313 Table 67 – GetPDRRepositoryInfo command format 120

314 Table 68 – GetPDR command format..... 121

315 Table 69 – FindPDR command format 125

316 Table 70 – FindPDR Command Parameter Format Numbers 128

317 Table 71 – FindPDR command parameter formats 129

318 Table 72 – RunInitAgent command format 130

319 Table 73 – GetPDRRepositorySignature command format..... 131

320 Table 74 – sensorUnits enumeration 133

321 Table 75 – Common PDR header format 142

322 Table 76 – PDR Type Values..... 144

323 Table 77 – Terminus Locator PDR format 145

324 Table 78 – Numeric Sensor PDR format 147

325 Table 79 – Numeric Sensor Initialization PDR format 153

326 Table 80 – State Sensor PDR format 154

327 Table 81 – State Sensor possible states fields format..... 155

328 Table 82 – State Sensor Initialization PDR format 156

329 Table 83 – Sensor Auxiliary Names PDR format..... 159

330 Table 84 – OEM Unit PDR format..... 160

331 Table 85 – OEM State Set PDR format 162

332 Table 86 – OEM State Value Record format 163

333 Table 87 – Numeric Effector PDR format 164

334 Table 88 – Numeric Effector Initialization PDR format 168

335 Table 89 – State Effector PDR format 169

336 Table 90 – State Effector Possible States fields format..... 170

337 Table 91 – State Effector Initialization PDR format 171

338 Table 92 – Effector Auxiliary Names PDR format..... 173

339 Table 93 – OEM Effector Semantic PDR format..... 174

340 Table 94 – Entity Association PDR format..... 175

341 Table 95 – Entity Auxiliary Names PDR format 176

342 Table 96 – OEM EntityID PDR format 177

343 Table 97 - Interrupt Association PDR format 178

344 Table 98 – Event Log PDR format 179

345 Table 99 – FRU Record Set PDR format..... 181

346 Table 100 – OEM Device PDR format 182

347 Table 101 – OEM PDR format 182

348 Table 102 – Compact Numeric Sensor PDR format..... 183

349 Table 103 – Redfish Resource PDR format???. 185

350 Table 104 – Redfish Entity Association PDF format???. 188

351 Table 105 – Redfish Action PDR format???. 189

352 Table 106 – Monitoring and control timing specifications 190

353 Table 107 – Command numbers 190

354

355

356

Foreword

357 The *Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification* (DSP0248) was
358 prepared by the Platform Management Components Intercommunications (PMCI) Working Group of the
359 DMTF.

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

362 Acknowledgments

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

364 Editors:

- 365 • Patrick Schoeller and Bill Scherer – Hewlett Packard Enterprise

366 Contributors:

- 367 • Richelle Ahlvers – Broadcom Inc.
- 368 • Alan Berenbaum – SMSC
- 369 • Chris Bussan – Hewlett Packard Enterprise
- 370 • Patrick Caporale – Lenovo
- 371 • Phil Chidester – Dell
- 372 • Hoan Do – Broadcom Inc.
- 373 • Yuval Itkin – Mellanox Technologies
- 374 • Ed Klodnicki – IBM
- 375 • John Leung – Intel Corporation
- 376 • Eliel Louzoun – Intel Corporation
- 377 • Balaji Natrajan – Microsemi
- 378 • Hemal Shah – Broadcom Inc.
- 379 • Tom Slaight – Intel Corporation
- 380 • Bob Stevens – Dell
- 381 • Supreeth Venkatesh – Arm Limited

382

383

Introduction

384 The *Platform Level Data Model (PLDM) Monitoring and Control Specification* defines messages and data
385 structures for discovering, describing, initializing, and accessing sensors and effecters within the
386 management controllers and management devices of a platform management subsystem. Additional
387 functions related to platform monitoring and control, such as the generation and logging of platform level
388 events, are also defined.

389 **Document conventions**

390 **Typographical conventions**

391 The following typographical conventions are used in this document:

- 392 • Document titles are marked in *italics*.
 - 393 • Important terms that are used for the first time are marked in *italics*.
- 394

396
397

Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification

1 Scope

399 This specification defines the functions and data structures used for discovering, describing, initializing,
400 and accessing sensors and effecters within the management controllers and management devices of a
401 platform management subsystem using PLDM messaging. Additional functions related to platform
402 monitoring and control, such as the generation and logging of platform level events, are also defined. This
403 document does not specify the operation of PLDM messaging.

404 This specification is not a system-level requirements document. The mandatory requirements stated in
405 this specification apply when a particular capability is implemented through PLDM messaging in a manner
406 that is conformant with this specification. This specification does not specify whether a given system is
407 required to implement that capability. For example, this specification does not specify whether a given
408 system must provide sensors or effecters. However, if a system does implement sensors or effecters or
409 other functions described in this specification, the specification defines the requirements to access and
410 use those functions under PLDM.

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

- 413 • DMTF [DSP0240](#), *Platform Level Data Model (PLDM) Base Specification*, provides definitions of
414 common terminology, conventions, and notations used across the different PLDM specifications
415 as well as the general operation of the PLDM messaging protocol and message format.
- 416 • DMTF [DSP0249](#), *Platform Level Data Model (PLDM) State Sets Specification*, defines the
417 values that are used to represent different types of states and entities within this specification.

2 Normative references

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

423 ANSI/IEEE Standard 754-1985, *Standard for Binary Floating Point Arithmetic*

424 DMTF DSP0218 *Platform Level Data Model for Redfish Device Enablement 1.0*
425 http://dmtof.org/sites/default/files/standards/documents/DSP0218_1.0.pdf

426 DMTF DSP0236, *MCTP Base Specification 1.0*,
427 http://dmtof.org/sites/default/files/standards/documents/DSP0236_1.0.pdf

428 DMTF DSP0240, *Platform Level Data Model (PLDM) Base Specification 1.0*,
429 http://dmtof.org/sites/default/files/standards/documents/DSP0240_1.0.0.pdf

430 DMTF DSP0241, *Platform Level Data Model (PLDM) Over MCTP Binding Specification 1.0*,
431 http://dmtof.org/sites/default/files/standards/documents/DSP0241_1.0.pdf

432 DMTF DSP0245, *Platform Level Data Model (PLDM) IDs and Codes Specification 1.0*,
433 http://dmtof.org/sites/default/files/standards/documents/DSP0245_1.0.pdf

- 434 DMTF DSP0249, *Platform Level Data Model (PLDM) State Sets Specification 1.0*,
435 http://dmtof.org/sites/default/files/standards/documents/DSP0249_1.0.pdf
- 436 DMTF DSP0257, *Platform Level Data Model (PLDM) FRU Data Specification 1.0*,
437 http://dmtof.org/sites/default/files/standards/documents/DSP0257_1.0.pdf
- 438 DMTF DSP0266, *Redfish Scalable Platforms Management API Specification 1.6.0*,
439 https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.6.0.pdf
- 440 IETF RFC2781, *UTF-16, an encoding of ISO 10646*, February 2000,
441 <http://www.ietf.org/rfc/rfc2781.txt>
- 442 IETF RFC3629, *UTF-8, a transformation format of ISO 10646*, November 2003,
443 <http://www.ietf.org/rfc/rfc3629.txt>
- 444 IETF RFC4122, *A Universally Unique Identifier (UUID) URN Namespace*, July 2005,
445 <http://www.ietf.org/rfc/rfc4122.txt>
- 446 IETF RFC4646, *Tags for Identifying Languages*, September 2006,
447 <http://www.ietf.org/rfc/rfc4646.txt>
- 448 ISO 8859-1, *Final Text of DIS 8859-1, 8-bit single-byte coded graphic character sets — Part 1: Latin*
449 *alphabet No. 1*, February 1998
- 450 ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,
451 <http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype>

452 **3 Terms and definitions**

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

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

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

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

466 Refer to [DSP0240](#) for terms and definitions that are used across the PLDM specifications. For the
467 purposes of this document, the following additional terms and definitions apply.

468 **3.1**

469 **contained entity**

470 an entity that is contained within a container entity

471 **3.2**

472 **container entity**

473 an entity that is identified as containing or comprising one or more other entities

- 474 **3.3**
475 **container ID**
476 a numeric value that is used within Platform Descriptor Records (PDRs) to uniquely identify a container
477 entity
- 478 **3.4**
479 **containing entity**
480 an alternative way of referring to the container entity for a given entity
- 481 **3.5**
482 **entity**
483 a particular physical or logical entity that is identified using PLDM monitoring and control data structures
484 for the purpose of monitoring, controlling, or identifying that entity within the platform management
485 subsystem, or for identifying the relationship of that entity to other entities that are monitored or controlled
486 using PLDM monitoring and control
487 Examples of physical entities include processors, fans, power supplies, and memory chips. Examples of
488 logical entities include a logical power supply (which may comprise multiple physical power supplies) and
489 a logical cooling unit (which may comprise multiple fans or cooling devices).
- 490 **3.6**
491 **Entity ID**
492 a numeric value that is used to identify a particular type of entity, but without designating whether that
493 entity is a physical or logical entity
- 494 **3.7**
495 **Entity Instance Number**
496 a numeric value that is used to differentiate among instances of the same type
497 For example, if two processor entities exist, one of them can be designated with instance number 1 and
498 the other with instance number 2.
- 499 **3.8**
500 **Entity Type**
501 a numeric value that identifies both the particular type of entity and whether the entity is a physical or
502 logical entity
503 The Entity ID is a subfield of the Entity Type.
- 504 **3.9**
505 **Platform Descriptor Record**
506 **PDR**
507 a set of data that is used to provide semantic information about sensors, effecters, monitored or controller
508 entities, and functions and services within a PLDM implementation
509 PDRs are mostly used to support PLDM monitoring and control and platform events. This information also
510 describes the relationships (associations) between sensor and control functions, the physical or logical
511 entities that are being monitored or controlled, and the semantic information associated with those
512 elements.

513 **4 Symbols and abbreviated terms**

514 Refer to [DSP0240](#) for symbols and abbreviated terms that are used across the PLDM specifications. For
 515 the purposes of this document, the following additional symbols and abbreviated terms apply.

516 **4.1**

517 **CIM**

518 Common Information Model

519 **4.2**

520 **EID**

521 Endpoint ID

522 **4.3**

523 **IANA**

524 Internet Assigned Numbers Authority

525 **4.4**

526 **MAP**

527 Manageability Access Point

528 **4.5**

529 **MCTP**

530 Management Component Transport Protocol

531 **4.6**

532 **PDR**

533 Platform Descriptor Record

534 **4.7**

535 **PLDM**

536 Platform Level Data Model

537 **4.8**

538 **TID**

539 Terminus ID

540 **5 Conventions**

541 Refer to [DSP0240](#) for conventions, notations, and data types that are used across the PLDM
 542 specifications. The following data types are also defined for use in this specification:

543 **Table 1 – PLDM monitoring and control data types**

Data type	Interpretation
strASCII	A null (0x00) terminated 8-bit per character string. Unless otherwise specified, characters are encoded using the 8-bit ISO8859-1 "ASCII + Latin1" character set encoding. All strASCII strings shall have a single null (0x00) character as the last character in the string. Unless otherwise specified, strASCII strings are limited to a maximum of 256 bytes including null terminator.

Data type	Interpretation
strUTF-8	A null (0x00) terminated, UTF-8 encoded string per RFC3629 . UTF-8 defines a variable length for Unicode encoded characters where each individual character may require one to four bytes. All strUTF-8 strings shall have a single null character as the last character in the string with encoding of the null character per RFC3629 . Unless otherwise specified, strUTF-8 strings are limited to a maximum of 256 bytes including null terminator character.
strUTF-16	A null (0x0000) terminated, UTF-16 encoded string with Byte Order Mark (BOM) per RFC2781 . All strUTF-16 strings shall have a single null (0x0000) character as the last character in the string. An empty string shall be represented using two bytes set to 0x0000, representing a single null (0x0000) character. Otherwise, the first two bytes shall be the BOM. Unless otherwise specified, strUTF-16 strings are limited to a maximum of 256 bytes including the BOM and null terminator.
strUTF-16LE	A null (0x0000) terminated, UTF-16, "little endian" encoded string per RFC2781 . All strUTF-16LE strings shall have a single null (0x0000) character as the last character in the string. Unless otherwise specified, strUTF16LE strings are limited to a maximum of 256 bytes including the null terminator.
strUTF-16BE	A null (0x0000) terminated, UTF-16, "big-endian" encoded string per RFC2781 . All strUTF-16BE strings shall have a single null character as the last character in the string. Unless otherwise specified, strUTF16BE strings are limited to a maximum of 256 bytes including the null terminator.

544 6 PLDM for Platform Monitoring and Control version

545 The version of this *Platform Level Data Model (PLDM) for Platform Monitoring and Control Specification*
 546 shall be 1.2.0 (major version number 1, minor version number 2, update version number 0, and no alpha
 547 version).

548 For the GetPLDMVersion command described in [DSP0240](#), the version of this specification is reported
 549 using the encoding as 0xF1F2F000.

550 If the endpoint declares support for PLDM for Platform Monitoring and Control version 1.1.1 or later
 551 specification versions, all previous versions (e.g., 1.1.0) should not be listed as supported in the
 552 GetPldmVersion command because of the sensorID (Numeric Sensor PDR) or the effectorID (Numeric
 553 Effector PDR) size change from uint8 to uint16.

554 7 PLDM for Platform Monitoring and Control overview

555 This specification describes the operation and format of request messages (also referred to as
 556 commands) and response messages for accessing the monitoring and control functions within the
 557 management controllers and management devices of a platform management subsystem. These
 558 messages are designed to be delivered using PLDM messaging.

559 The basic format that is used for sending PLDM messages is defined in [DSP0240](#). The format that is
 560 used for carrying PLDM messages over a particular transport or medium is given in companion
 561 documents to the base specification. For example, [DSP0241](#) defines how PLDM messages are formatted
 562 and sent using MCTP as the transport. The *Platform Level Data Model (PLDM) for Platform Monitoring*
 563 *and Control Specification* defines messages that support the following items:

- 564 • sensors and effecters

565 This specification defines a model for sensors and effecters through which monitoring and
 566 control are achieved, and the commands that are used for sensor and effector initialization,
 567 configuration, and access. Sensors and effecters are classified according to the general type of
 568 data that they use:

- 569 – Numeric sensors provide a number that represents a monitored value that can be
570 expressed using units such as degrees Celsius, volts, and amps.
- 571 – State sensors are used for accessing a number from an enumeration that represents the
572 state of a monitored entity. Different states are enumerated in predefined sets called state
573 sets. Example state sets can include states for Availability (enabled, disabled, shut down,
574 and so on), Door State (open, closed), Presence (present, not present) and so on. The
575 values for State Sets are defined in [DSP0249](#).
- 576 – Numeric effecters are used for setting a number that configures or controls the operation of
577 a controlled entity. Like numeric sensors, numeric effecters also use units such as degrees
578 Celsius, volts, and amps.
- 579 – State effecters are used for setting a number that configures or controls a state that is
580 associated with a controlled entity. State effecters draw upon the same state set definitions
581 as state sensors.
- 582 • Platform Descriptor Records (PDRs)
- 583 PDRs are data structures that can provide semantic information for sensors and effecters, their
584 relationship to the entities that are being monitored or controlled, and associations that exist
585 between entities within the platform. The PDRs also include information that describes the
586 presence and location of different PLDM termini. This information can be used to discover the
587 population of sensors and effecters and how to access them by using PLDM messaging. The
588 information also facilitates building Common Information Model objects and associations for the
589 sensors, effecters, and platform entities. PDRs can also hold information that is used to initialize
590 sensors and effecters. PDRs are collected into a logical storage area called a PDR Repository.
591 A central PDR Repository called the Primary PDR Repository can be used to hold an
592 aggregation of all PDR information within the PLDM subsystem.
- 593 • platform events
- 594 This specification defines messages that are asynchronously sent upon particular state changes
595 that occur within sensors, effecters, or the PLDM platform management subsystem. The
596 messages are delivered to a central function called the PLDM Event Receiver. Version 1.2.0 of
597 this specification also defines a synchronous polling method to retrieve events from an entity.
- 598 • platform event logging
- 599 The specification includes the definition of a central, nonvolatile storage function called the
600 PLDM Event Log that can be used to log PLDM Event Messages. The specification also defines
601 messages for accessing and maintaining the PLDM Event Log.
- 602 • support functions
- 603 This specification also includes the definition of support functions as required to support the
604 initialization of sensors and effecters, and the maintenance of PDRs in the Primary PDR
605 Repository. The main support functions are the Discovery Agent and the Initialization Agent.
- 606 – The Discovery Agent function is responsible for keeping the Primary PDR information up to
607 date if entities are added, relocated, or removed from the PLDM platform management
608 subsystem. The Discovery Agent function is also responsible for setting the Event Receiver
609 location into PLDM termini that support PLDM monitoring and control messages.
- 610 – The Initialization Agent function is responsible for initializing sensors and effecters that may
611 require initialization or reinitialization upon state changes to the PLDM terminus or the
612 managed system, such as system hard resets, the terminus coming online for PLDM
613 communication, and so on.
- 614 • OEM/vendor-specific functions
- 615 This specification includes provisions for supporting OEM or vendor-specific functions and
616 semantic information. This includes the ability to define OEM units for numeric sensors or

617 effecters, OEM state sets, and OEM entity types. An OEM PDR type is also available as an
618 opaque storage mechanism for holding OEM-defined data in PDR Repositories.

619 **8 PDR architecture**

620 This clause provides an overview of when and how PDRs are used within a platform management
621 subsystem that uses the PLDM Platform Monitoring and Control commands.

622 **8.1 General**

623 PLDM generally separates the access of functions such as sensors and effecters from the semantic
624 information or description of those functions. For example, PLDM commands such as
625 GetNumericSensorReading return binary values for a sensor, but the meaning of those values, such as
626 whether they represent a temperature or voltage, is described separately. The description or semantic
627 information for sensors, effecters, and other elements of the PLDM platform management subsystem is
628 provided through Platform Descriptor Records, or PDRs.

629 This separation provides several benefits:

- 630 • Overhead for simple Intelligent Management Devices is reduced. In many implementations, a
631 primary management controller may access one or two simpler controllers that act as Intelligent
632 Management Devices (sometimes also called "satellite controllers"). Those controllers generally
633 are very cost sensitive and limited in resources such as RAM, nonvolatile storage capabilities,
634 data transfer performance, and so on. The amount of data that needs to be stored and
635 transferred to provide the semantic information for a sensor is typically an order of magnitude or
636 more greater than the amount of data that needs to be transferred to get the state or reading
637 information from a sensor.
- 638 • PDRs provide information that associates sensors, effecters, and the entities that are being
639 monitored or controlled within the overall context of the PLDM platform management
640 subsystem. This eliminates the need for devices that implement sensors and effecters to
641 understand their position and use in the overall system. Providing this association and context
642 information for sensors and effecters enables the automatic instantiation of CIM objects and
643 CIM associations.
- 644 • The impact of extensions to descriptions is reduced. The definitions of the semantic information
645 (PDRs) can be extended and modified without affecting the commands that are used to access
646 sensors and effecters.

647 **8.2 Primary PDR Repository and Device PDR repositories**

648 The PDRs for a PLDM subsystem are collected into a single, central PDR Repository called the Primary
649 PDR Repository. A central repository provides a single place from which PDR information can be
650 retrieved and simplifies the inter-association of PDR semantic information for the different elements and
651 monitored or controlled entities within the subsystem.

652 Individual devices, such as hot-plug devices, can hold their own Device PDRs that describe their local
653 semantics. Typically, this information has only local context. That is, the information covers only the
654 elements on the add-in card and has no information about the positioning of the card and its capabilities
655 relative to the overall subsystem. Thus, additional steps are typically taken to integrate Device PDR
656 information into the overall context of the PLDM subsystem.

657 **8.3 Use of PDRs**

658 Whether PDRs are used is based on the needs and goals of the PLDM subsystem implementation. This
659 subclause describes three different applications of PLDM and their level of PDR support.

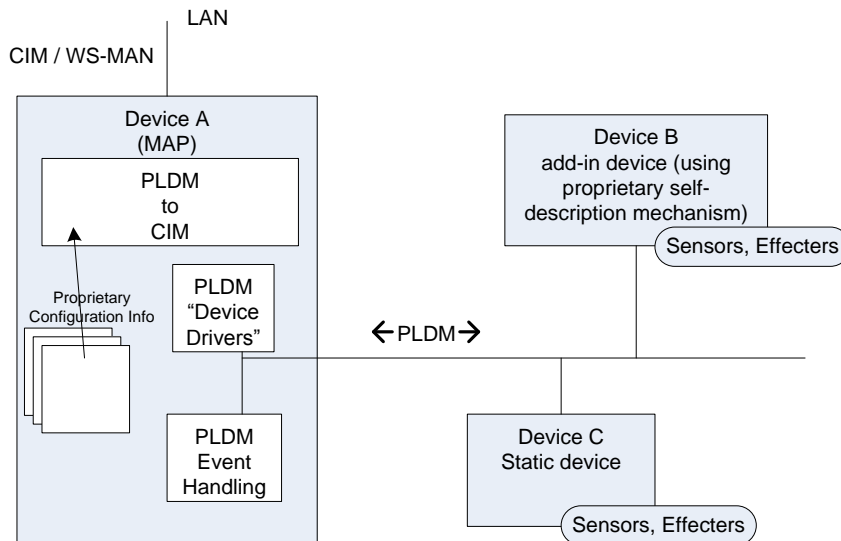
660 **8.3.1 PLDM for access only**

661 Figure 1 shows an implementation that does not use PDRs. PLDM is used only as a mechanism for
 662 accessing monitoring and control functions; it is not used for providing semantic information about those
 663 functions.

664 In this example, Device A provides a DMTF Manageability Access Point (MAP) function that makes
 665 platform information available over a network using CIM as the data model and WS-MAN as the transport
 666 protocol for CIM. In this example, PLDM is used only for accessing the functions in Devices B and C, and
 667 for Devices B and C to send PLDM Event Messages to Device A.

668 All the semantic or descriptive information that is needed to map the sensors and effecters to CIM objects
 669 and properties is handled by proprietary mechanisms. Typically a vendor-specific configuration utility is
 670 used by the system integrator to configure or customize a set of proprietary configuration information that
 671 provides whatever contextual or semantic information is required for the particular platform
 672 implementation. Since the mechanisms for recording semantic information are proprietary, most of the
 673 PLDM-to-CIM mapping function is also proprietary. A standard approach for the PLDM-to-CIM mapping
 674 function cannot be specified when proprietary mechanisms are used for the semantic information.

675 Thus, in this example PLDM does not offer much to assist or direct the way sensor and effector functions
 676 of external management devices would be mapped into the instantiation of CIM objects. The
 677 implementation only uses PLDM to provide a common mechanism for accessing the functions in the
 678 external Intelligent Management Devices. This enables the implementation to be designed with Device
 679 Driver and PLDM Event Handling code that can be reused if it is necessary to change the design to
 680 support different external Intelligent Management Devices.



681

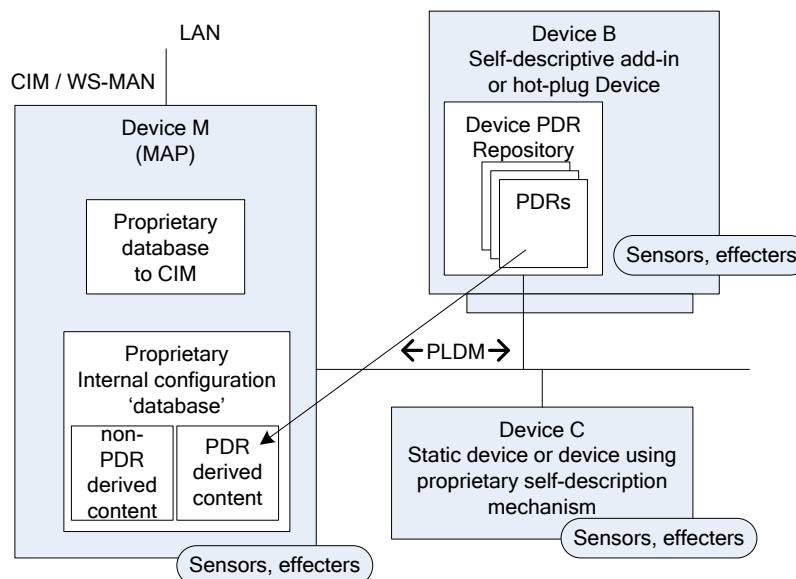
682 **Figure 1 – PLDM used for access only**

683 **8.3.2 PLDM with PDRs for add-in devices**

684 Figure 2 illustrates how PDRs can be used with add-in cards. The vendor of an add-in card knows the
 685 relationships and semantics of the monitoring and control (sensor and effector) capabilities on their card.

686 However, the vendor of the card typically will not know the relationship that card will have relative to a
 687 particular overall system. For example, the vendor would not know a priori what the system name was, or
 688 how many processors the system has, or into which slot the card will be plugged. Thus, in this example,
 689 the add-in card exports PDRs that describe the relationships relative to the add-in card. The MAP takes
 690 this information and integrates it into the semantic view of the overall system. The PDR information could
 691 be converted and linked into a proprietary internal database, as shown in Figure 2. The PDRs thus
 692 provide a common way for add-in cards to describe themselves to the MAP.

693 The internal database for the MAP could be implemented as a PDR Repository instead of a proprietary
 694 database. This would potentially simplify the PLDM-to-CIM mapping process, enabling the integrated data
 695 to be accessed as PDRs using PDR Repository access commands and enabling software or other parties
 696 to see the integrated view of the platform at the PLDM level. Also, because the PLDM-to-CIM mapping is
 697 defined using PDRs, the PDR format may also be useful in developing a consistent PLDM-to-CIM
 698 mapping in the MAP.



699

700

Figure 2 – PLDM with device PDRs

701 **8.3.3 PLDM with Primary PDR Repository**

702 Figure 3 shows an example of using PDRs to describe an entire PLDM platform management subsystem
 703 to an add-in card, Device M, that provides a MAP function. In this example, PDRs are collected into a
 704 central PDR Repository called the Primary PDR Repository that is provided by Device A.

705 The PDRs in the Primary PDR Repository represent the entire PLDM subsystem behind Device A. Thus,
 706 the MAP of Device M needs to connect only to Device A to discover and get semantic information about
 707 the monitoring and control functions for that entire subsystem. This approach can enable Device M to
 708 automatically adapt itself to the management capabilities offered by different systems.

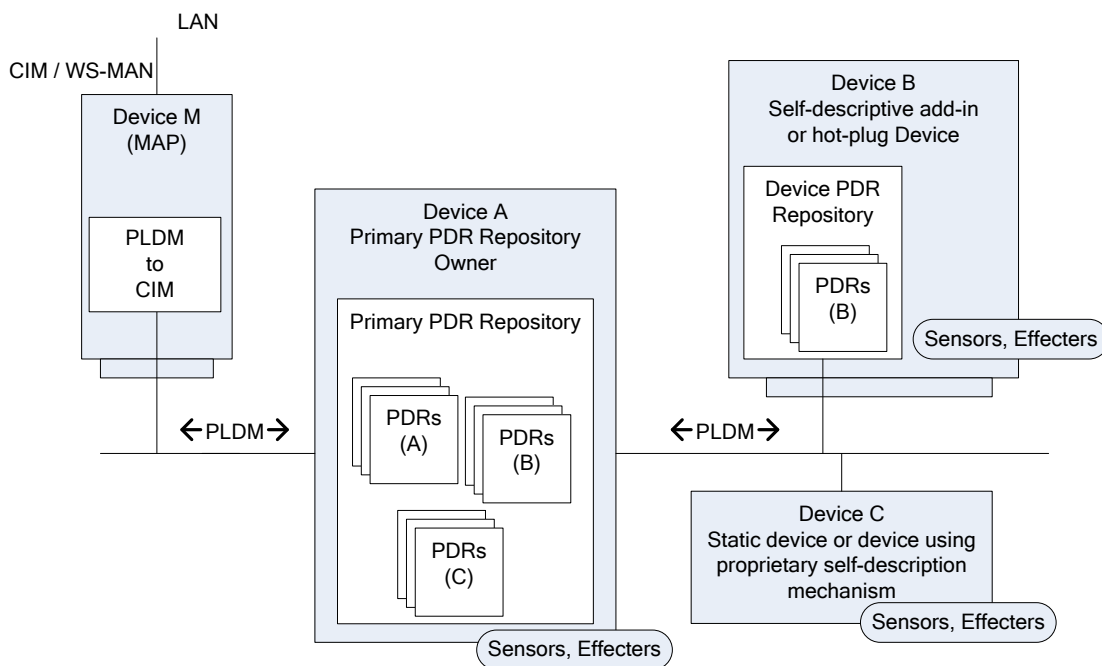
709 Such an implementation enables the MAP to come from one party while the platform management
 710 subsystem comes from another without the need to explicitly configure the MAP with the semantic
 711 information for the subsystem. For example, the platform management subsystem represented through

712 Device A could be built into a motherboard and the MAP of Device M provided on a PCIe add-in card
 713 from a third party. The MAP on the add-in card can use the Primary PDR Repository to automatically
 714 discover the capabilities and semantic information of the platform management subsystem and use that
 715 information to instantiate CIM objects and data structures for the subsystem.

716 Device A maintains the Primary PDR Repository that includes information about static sensors and
 717 effecters (such as those within Device C and within Device A itself) and integrates that information into
 718 the overall view of the platform management subsystem held in the Primary PDR Repository. This
 719 involves discovering and extracting PDRs from "Self-descriptive" devices such as Device B, and
 720 synthesizing additional PDRs, such as association and Terminus Locator PDRs, in order to integrate the
 721 PDRs into the repository and create a coherent view of the overall subsystem.

722 Because Device M is an add-in card, it could also have its own sensors and effecters and associated
 723 PDRs that Device A would integrate into the Primary PDR Repository in the same manner that it
 724 integrates PDR information from Device B.

725 Another advantage of implementing a Primary PDR Repository is that any party with access to Device A
 726 can get the full set of semantic information for the subsystem. This is useful when more than one party
 727 might need to access that information—for example, if support was necessary for multiple add-in cards
 728 that provided MAP functions for different media (such as one card that provided MAP functions over
 729 cabled Ethernet and another that provided MAP access using a wireless network connection).



730

731

Figure 3 – PLDM with PDRs for subsystem

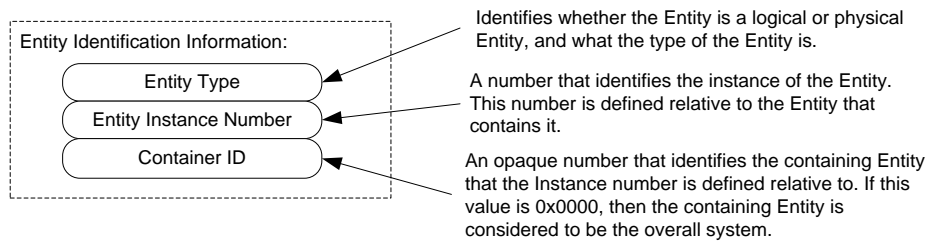
732 **9 Entities**

733 Within the context of this specification, the term entity is used to refer to either a physical or a logical
 734 entity that is monitored or controlled, or to describe the topology or structure of the system that is being
 735 monitored or controlled.

736 Examples of typical physical entities include processors, fans, memory devices, and power supplies.
 737 Examples of logical entities include logical power supplies that are formed from multiple physical power
 738 supplies (as in the case of a redundant power supply subsystem) and a logical cooling unit formed from
 739 multiple physical fans.

740 **9.1 Entity Identification Information**

741 Individual entities are identified within PLDM PDRs using three fields: Entity Type, Entity Instance
 742 Number, and Container ID. Together, these fields are referred to as the Entity Identification Information.
 743 Figure 4 presents an overview of the meaning of the individual fields. The fields are discussed in more
 744 detail in the next subclauses.



745

746 **Figure 4 – Entity Identification Information**

747 The combination of Entity Type, Entity Instance Number, and Container ID must be unique for each
 748 individual entity referenced in the PDRs. These three fields are always used together in the PDRs and in
 749 the same order. The combination of the three fields is represented in the PDRs using three uint16 values
 750 in the format shown in Figure 5.

15	14	0	
P/L	Entity ID		Entity Type
Entity Instance Number			
Container ID			

751

752 **Figure 5 – Entity Identification Information format**

753 Table 2 describes the parts of the Entity Identification Information format.

754 **Table 2 – Parts of the Entity Identification Information format**

Part	Description
Entity Type	Combination of the P/L bit and the Entity ID value
P/L	Physical/Logical bit (0b = physical, 1b = logical)
Entity ID	15-bit Entity ID value from DSP0249 that identifies the general type of the entity
Entity Instance Number	16-bit number that differentiates among instances of entities that have the same Entity Type and Container ID values
Container ID	16-bit number that identifies the containing entity that the Entity Instance Number is defined relative to. If this value is 0x0000, the containing entity is considered to be the overall system.

755 9.2 Entity Type and Entity IDs

756 The Entity Type field is a concatenation of the physical/logical designation for the entity and the value
 757 from the Entity ID enumeration that identifies the general type or category of the entity, such as whether
 758 the entity is a power supply, fan, processor, and so on. The Entity Type field indicates whether the entity
 759 is a physical fan, logical power supply, and so on.

760 The different general types of entities within PLDM are identified using an enumeration value referred to
 761 as an "Entity ID." The different types of standardized entities and their corresponding Entity ID values are
 762 specified in [DSP0249](#).

763 Physical and logical entities that have the same Entity ID are considered to be different Entity Types.

764 9.2.1 Vendor-specific (OEM) Entity IDs

765 The Entity ID values include a special range of values for identifying vendor- or OEM-specific entities. In
 766 order to be interpreted, these values must be accompanied by an OEM EntityID PDR that identifies which
 767 vendor defined the entity and, optionally, a string or strings that provide the name for the entity. Refer to
 768 28.19 for additional information about how OEM Entity IDs are used.

769 9.2.2 Logical and physical entities

770 A physical entity is defined as an entity that is formed from one or more physically identifiable
 771 components. For example, a physical Power Supply could be one or more integrated circuits and
 772 associated components that together form a power supply.

773 A logical entity is defined as an entity that is formed when the entity or grouping of entities lacks a
 774 physical definition or a readily identifiable physical boundary or grouping that would be associated with
 775 the type of entity being represented. For example, a logical cooling device could be used to represent a
 776 combination of physical fans that forms a redundant fan subsystem, or a logical power supply could be
 777 used to represent the combination or grouping of power supplies that forms a redundant power supply
 778 subsystem.

779 The choice of when to use a logical or physical designation for a particular type of entity can be subtle.
 780 Consider the following questions:

- 781 • Is the entity or grouping of entities separately replaceable or identifiable as a single physical unit
 782 or as a set of physical units?
- 783 • Would the physical grouping be something that a user would typically think of as a separate
 784 physical unit that can be represented by a single type of entity?

785 For example, consider a system with a motherboard that directly supports connectors for a redundant fan
786 configuration. The fans would typically be individually replaceable, and the motherboard would be
787 individually replaceable, but the "redundant fan subsystem" would not be. A user would not typically
788 consider the combination of a motherboard and fans to be the definition of a physical redundant fan
789 subsystem because the motherboard provides many other functions beyond those that are part of the
790 implementation of a redundant fan subsystem. The redundant fan subsystem does not have a distinct
791 physical boundary that would let it be replaced independently from other subsystems.

792 **9.3 Entity Instance Numbers**

793 A given platform often has more than one occurrence of a particular type of entity. The Entity Instance
794 Number, in combination with the Container ID, differentiates one instance of a particular type of entity
795 from another within the PDRs.

796 Entity Instance Numbers are defined in a numeric space that is associated with a particular containing
797 entity. For example, the Entity Instance Numbers for processors contained on an add-in card are defined
798 relative to that add-in card, whereas the Entity Instance Numbers for processors on the motherboard are
799 defined relative to the motherboard.

800 The Entity Instance Number is a value that could be used when instantiating CIM objects or presenting
801 PLDM data as part of the "name" of the managed object. For example, if a processor entity has an Entity
802 Instance Number of "1", the expectation is that the entity would be presented as "Processor 1".

803 The assignment of Entity Instance Number values under a given Container ID is left up to the
804 implementation. However, it is typical that Entity Instance Number values are allocated sequentially
805 starting from 0 or 1 for a given Entity Type under the Container ID.

806 **9.4 Container ID**

807 The value in this field identifies a "containing Entity" that in turn defines the numeric space under which
808 Entity Instance Numbers are allocated. For example, if an add-in card has two processors on it and a
809 motherboard has two processors on it, it would be common to refer to the processors on the add-in card
810 as "Processor 1" and "Processor 2" and to the processors on the motherboard also as "Processor 1" and
811 "Processor 2".

812 The Container ID field provides a mechanism that locates a particular containing entity, such as
813 "motherboard 1" or "add-in card 1". This enables the Entity Instance Numbers to be allocated relative to
814 each particular containing Entity. The Container ID field, therefore, effectively provides a value that
815 indicates that the "Processor 1" entity on the motherboard is a different entity than the "Processor 1"
816 entity on the add-in card.

817 In most cases, the Container ID field value points to a particular PDR that describes a "containment
818 association" that identifies a container entity (such as motherboard 1) and one or more contained entities
819 (such as processor 1 and processor 2). An exception occurs when an entity instance is defined only
820 relative to the overall system, in which case the Container ID holds a special value that indicates that the
821 "system" is the container entity.

822 **9.5 Use of Container ID in PDRs**

823 With the exception of the entity that represents an overall system, all entities are contained within at least
824 one other physical or logical entity. Each entity is thus part of a containment hierarchy that starts with the
825 overall system as the topmost entity. A strict hierarchy is formed when each entity is only allowed to
826 identify a single containing entity using the Container ID value. With this restriction, an entity's position in
827 the hierarchy can be uniquely identified, and when combined with the entity type and instance information
828 provides the unique Entity Identification Information for the entity. Thus, although a given entity may be
829 identified as being contained within more than one container entity, only one Container ID value shall be
830 used for the Entity Identification Information for an entity.

831 The Container ID points to a particular type of PDR called an Entity Association PDR that holds the
832 information that identifies and associates a containing entity with one or more contained entities.
833 Association PDRs are described in clause 10.

834 The overall system is considered to be the top of the hierarchy of containment and thus does not appear
835 as a contained entity in any Entity Association PDR. In this case, there is no explicit Entity Association
836 PDR for the overall system. A special value (0x0000) is used for the Container ID to indicate when the
837 overall system is the container entity.

838 In some cases, a particular entity may be part of more than one containment hierarchy. For example, a
839 physical fan could be part of a logical cooling unit *and* a physical chassis. When both physical and logical
840 containers exist for a given entity, the physical container relationship should be used for identifying the
841 entity.

842 **10 PLDM associations**

843 Different mechanisms are used to associate different elements of PLDM with one another. This clause
844 describes the different association mechanisms and how they're used.

845 **10.1 Association examples**

846 Following are some examples of associations that are covered by PDRs:

- 847 • Sensor/Effecter Semantic Information to Sensor/Effecter Access associations:
848 Sensor and effecter PDRs describe the characteristics of a particular sensor or effecter. These
849 records include information that can be used to identify which PLDM terminus provides the
850 interface to the sensor, and the parameters that are used to access that sensor. These records
851 provide a way to form an association between the semantic information for a sensor/effecter
852 (provided by other information in the PDRs) and the access of the sensor (provided by PLDM
853 commands for sensor or effecter access).
- 854 • Sensor/Effecter to Entity associations:
855 A sensor or effecter monitors or controls some physical or logical entity. The PDRs provide a
856 mechanism for associating a sensor or effecter with the entity.
- 857 • Entity to Entity associations:
858 Entities have relationships with other entities, such as physical and logical containment. For
859 example, a redundant power supply subsystem may be represented as a logical power supply
860 that is made up of multiple physical power supplies.
- 861 • PLDM Event to PDR associations:
862 PLDM Event Messages identify the terminus that was the source of the message, and the
863 sensor within the terminus that was the source of the event, but semantic information and the
864 context for the sensor are not carried in the event information. The PDRs include information
865 that associates the information in an event message with the semantic information that enables
866 interpretation of the event and its context.

867 Two general mechanisms are used for specifying associations for PLDM: Internal Associations and
868 External Associations.

869 **10.2 Internal and External Associations**

870 The term "Internal Association" is used when a particular type of association is formed solely by using
871 fields within the PDRs that directly associate PDRs with one another. For example, a value called the
872 Terminus Handle is used in all PDRs that are associated with a particular terminus. The Terminus Handle
873 is a form of Internal Association, where the association is "PDRs that belong to a given terminus." Internal
874 Associations effectively associate records by defining and using a common field as a key.

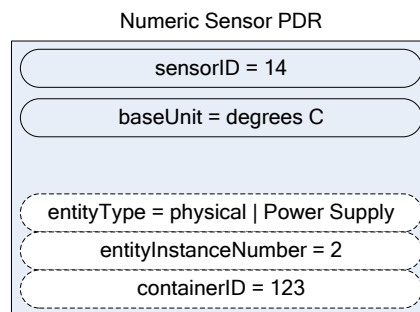
875 Therefore, Internal Associations require a common field to be defined among the elements that are
 876 associated with each other. The Internal Association mechanism is efficient, but not readily extensible,
 877 because a new type of association would typically require new fields to be defined and added to the
 878 PDRs that are to be associated with one another, along with specifications that document how the field is
 879 used to form links to other records. Because the fields that support Internal Associations must be pre-
 880 defined as part of the PDR, Internal Associations are generally used only for the most fundamental and
 881 common types of associations. For other types of associations, a more generalized mechanism called
 882 "External Associations" is provided.

883 External Associations are formed by using a separate data structure (PDR) to associate different
 884 elements with one another. This is accomplished among the PDRs by using another PDR that is referred
 885 to as an "association PDR." The advantage of using External Associations is that they enable
 886 associations between PDRs or entities without requiring the definition of common fields among them.
 887 Thus, new types of associations can be defined without requiring changes to existing PDR definitions.
 888 The disadvantage is that External Associations require the use of at least one additional PDR to form the
 889 association.

890 **10.3 Sensor/Effector to Entity associations**

891 Each sensor or effector that is described using PDRs has a corresponding Sensor or Effector PDR that
 892 provides semantic information for individual sensors or effectors, such as information that identifies which
 893 terminus the sensor or effector is associated with, the type of parameter that the sensor or effector is
 894 monitoring or controlling, and so on. Included in this information is Entity Identification Information for the
 895 entity that is associated with the sensor or effector. (The terms Sensor PDRs and Effector PDRs are used
 896 as shorthand to refer to a general class of PDRs. The actual PDRs define separate PDRs for numeric
 897 sensors, state sensors, numeric effectors, state effectors, and so on.)

898 Figure 6 shows a subset of the fields in the Sensor PDR for a PLDM Numeric Sensor. The Entity
 899 Identification Information is represented by the fields highlighted with dashed lines. Note that from this
 900 point in the document onward figures and tables will use field names as they are given in the definition of
 901 the PDRs, for example "entityInstanceNumber" instead of "entity instance number".



902

903 **Figure 6 – Entity Identification Information in a Numeric Sensor PDR**

904 Table 3 describes the meaning of the fields shown in Figure 6.

905 **Table 3 – Field & value descriptions for Entity Identification Information in a Numeric Sensor PDR**

Field and value	Description
sensorID = 14	All sensors and effecters within a given terminus have unique sensorID or effectorID numbers. This field holds a value that is used in commands such as GetSensorReading to access the particular sensor or effector within the terminus. The sensorID number is used only for accessing the sensor. The example shows that the value 14 would be used in commands to access this particular sensor.
baseUnit = degrees C	The baseUnit field identifies the measurement unit for the parameter being monitored by the sensor. The measurement unit is simplified for this example. The actual PDR contains additional fields that contribute to the definition of the measurement unit for a numeric sensor. Refer to the field's description in Table 78 for more information.
entityType = physical Power Supply	This field represents the concatenation of the physical/logical bit and the Entity ID for "power supply" from the Entity IDs table (see 9.2).
entityInstanceNumber = 2	The entityInstanceNumber differentiates instances of entities that have the same Entity Type and Container ID values. Because the entityInstanceNumber is defined relative to a containing entity, a system can have a processor on the motherboard identified as "processor 1" and a processor on an add-on card also identified as "processor 1". The two occurrences of "processor 1" are recognized as being unique and separate entities because they have different container entities. In this example, the entityInstanceNumber 2 indicates that this numeric sensor is monitoring physical Power Supply 2, which is contained within the container entity identified by containerID 123.
containerID = 123	This field is used to identify or locate the containing entity that defines the numeric space for the entityInstanceNumber. In this example, the number 123 would be used to locate an Entity Association PDR that identifies the containing entity (see 9.4 for more information). Association PDRs are described in detail in clause 11.

906 The details included in Table 3 provide a significant amount of the information that is typically used for
 907 identifying a sensor or effector and its use within a management subsystem. For example, a string that
 908 contains the following identification information for the sensor could be derived from the Numeric Sensor
 909 PDR without referring to any additional PDRs:

910 "Entity(123) physical power supply 2, Sensor(14), degrees C"

911 The information is based on the following fields:

912 container ID | entityType | entityInstanceNumber | sensorID | baseUnit

913 Note that an application would typically not use just the baseUnits name "degrees C" but would augment
 914 it to make it more readable. For example:

915 "Entity(123) physical power supply 2 Temperature Sensor(14) (Celsius)"

916 To interpret Entity(123), it is necessary to interpret the Container ID. If the Container ID is for "system,"
 917 the PDR may be interpreted as follows:

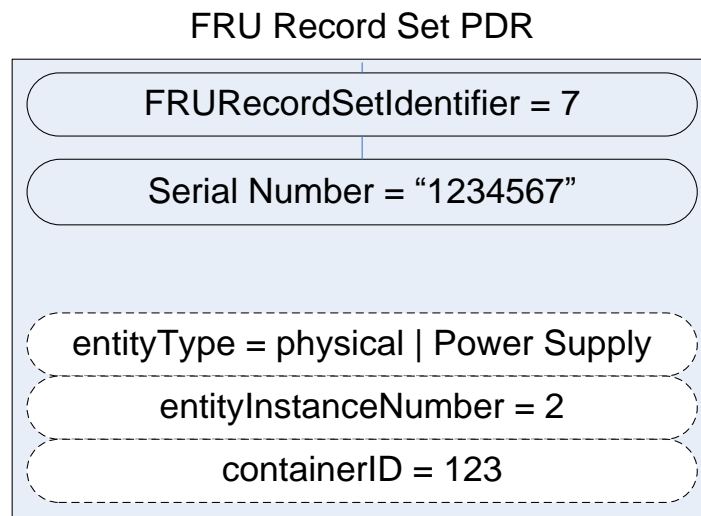
918 "System Physical Power Supply 2 Temperature Sensor (14) (Celsius)"

919 If the Container ID is for an entity other than system, the Container ID information can be used to locate
 920 the Entity Association PDR that identifies the containing entity for the sensor.

921 **10.4 FRU Record Set to Entity associations**

922 Each FRU Record Set that is described using PDRs has a corresponding FRU Record Set PDR that
 923 provides semantic information for individual FRUs, such as information that identifies which terminus is
 924 associated with the FRU Record Set. Included in this information is Entity Identification Information for the
 925 entity that is associated with the FRU Record Set.

926 Figure 7 shows a subset of the fields in the FRU Record Set PDR for a PLDM FRU Record Set. The
 927 Entity Identification Information is represented by the fields highlighted with dashed lines.



928

929 **Figure 7 – Entity Identification Information in a FRU Record Set PDR**

930 Table 4 describes the meaning of the fields shown in Figure 7.

931 **Table 4 – Field and value descriptions for Entity Identification Information in a FRU Record Set**
 932 **PDR**

Field and value	Description
FRURecordSetIdentifier = 7	All FRU Record Sets within a given terminus have unique Record Set Identifier. This field holds a value that is used in commands such as GetFRURecordByOption to access the particular Record Set within the terminus. The FRURecordSetIdentifier number is used only for accessing the FRU Record Set. The example shows that the value 7 would be used in commands to access this FRU Record Set.
Serial Number = "1234567"	The Serial Number field identifies the serial number of the FRU Record Set.
entityType = physical Power Supply	This field represents the concatenation of the physical/logical bit and the Entity ID for "power supply" from the Entity IDs table (see 9.2).
entityInstanceNumber = 2	The entityInstanceNumber differentiates instances of entities that have the same Entity Type and Container ID values. Because the entityInstanceNumber is defined relative to a containing entity, a system can have a processor on the motherboard identified as "processor 1" and a processor on an add-on card also identified as "processor 1". The two occurrences of "processor 1" are

Field and value	Description
	recognized as being unique and separate entities because they have different container entities. In this example, the entityInstanceNumber 2 indicates that this numeric sensor is monitoring physical Power Supply 2, which is contained within the container entity identified by containerID 123.
containerID = 123	This field is used to identify or locate the containing entity that defines the numeric space for the entityInstanceNumber. In this example, the number 123 would be used to locate an Entity Association PDR that identifies the containing entity (see 9.4 for more information). Association PDRs are described in detail in clause 11.

933 The details included in Table 4 provide a significant amount of the information that is typically used for
 934 identifying a FRU Record Set and its use within a management subsystem. For example, a string that
 935 contains the following identification information for the FRU Record Set could be derived from the FRU
 936 Record Set PDR without referring to any additional PDRs:

937 "Entity(123) physical power supply 2 Serial Number"

938 The information is based on the following fields:

939 container ID | entityType | entityInstanceNumber | Serial Number

940 Note that an application would typically use just Serial Number to make it more readable. For example:

941 "Entity(123) physical power supply 2 Serial Number"

942 To interpret Entity(123), it is necessary to interpret the Container ID. If the Container ID is for "system,"
 943 the PDR may be interpreted as follows:

944 "System Physical Power Supply 2 Serial Number"

945 If the Container ID is for an entity other than system, the Container ID information can be used to locate
 946 the Entity Association PDR that identifies the containing entity for the sensor.

947 11 Entity Association PDRs

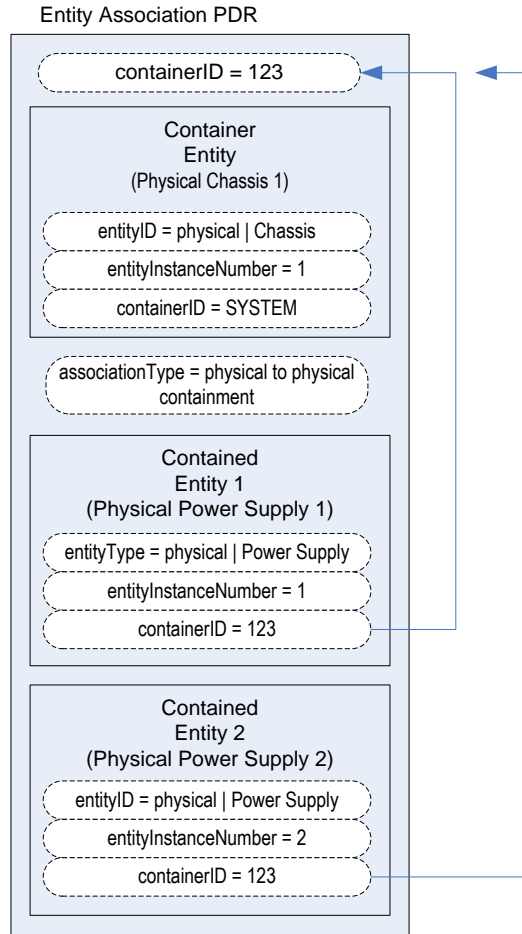
948 Entity Association PDRs associate entities with one another.

949 11.1 Physical-to-Physical containment associations

950 One of the most common associations is the "physical containment association." This association is used
 951 to indicate that a physical entity contains one or more other physical entities. For example, the
 952 association can be used to represent that a physical chassis contains multiple power supplies. Figure 8
 953 shows an example of selected fields within an Entity Association PDR that describes a physical
 954 containment association.

955 The example shows a containerID field and an associationType field in the PDR. The containerID is tied
 956 to the identification information for the container entity, which in this example is "system physical chassis
 957 1." The associationType field indicates that the association is a physical-to-physical containment
 958 association.

959 The record has entries for two contained power supplies: physical Power Supply 1 and physical Power
 960 Supply 2. The Entity Identification Information for both supplies refers back to the containerID 123 for the
 961 container entity, system physical chassis 1. Although this may appear redundant, it is done so that Entity
 962 Identification Information within PDRs is consistently represented with the same three-field format, and
 963 because in some types of associations the contained entity references the ID for a container entity that is
 964 identified in a different PDR.



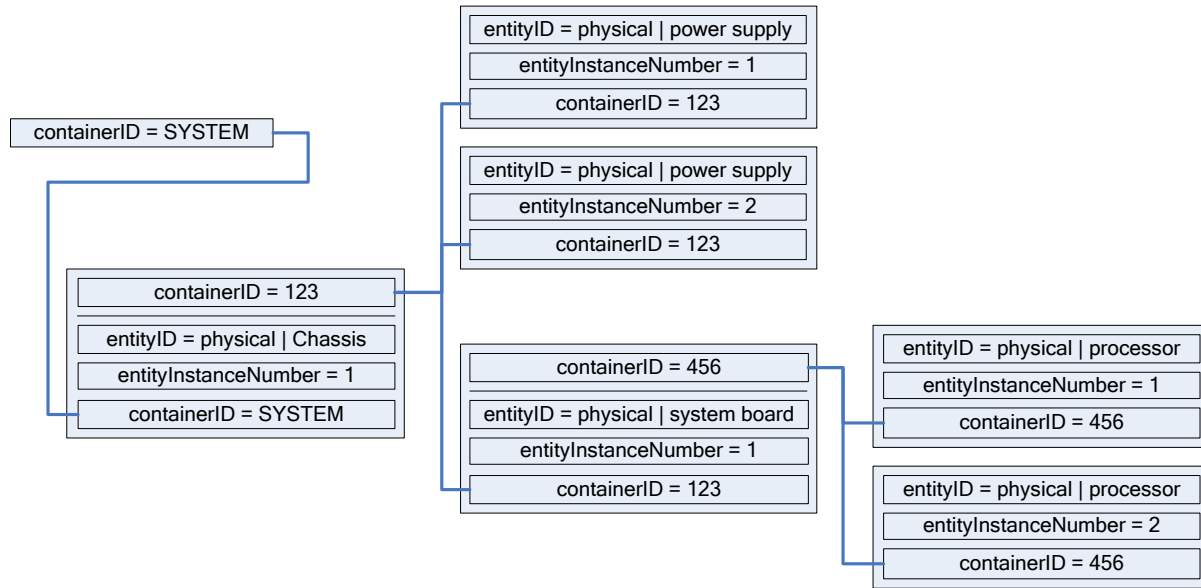
965

966

Figure 8 – Physical containment entity association PDR

967 Although the definition and use of the first containerID field might be confusing at first, think of the value
 968 as a single, unique number that identifies a container entity within the PLDM PDRs. The value thus
 969 represents the combination of the EntityType, entityInstanceNumber, and containerID values for the
 970 container entity. For example, referring to Figure 8, containerID 123 represents physical Chassis 1 (where
 971 instance number 1 is defined relative to SYSTEM).

972 Figure 9 provides an illustration of how the containerID value links entities in a containment hierarchy.



973

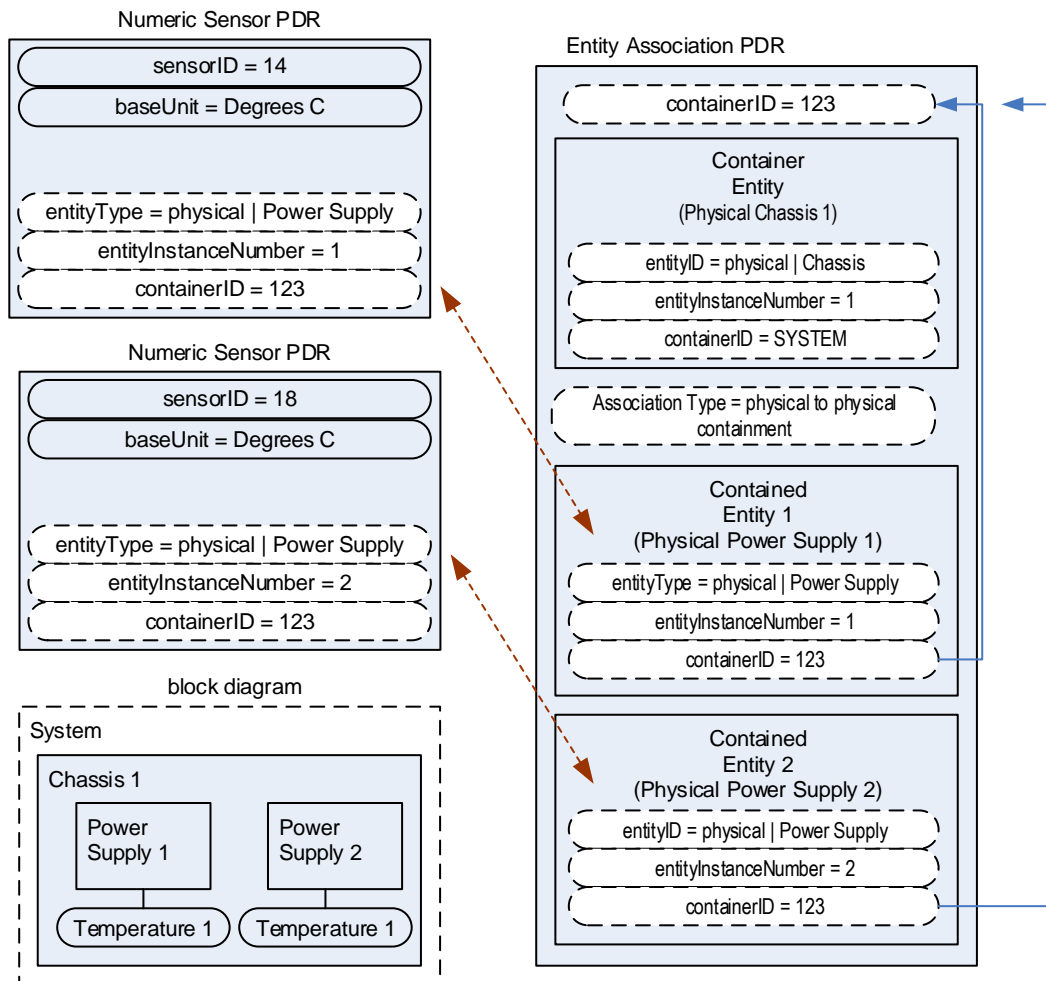
974

Figure 9 – containerID relationships

975 **11.2 Entity identification relationships between PDRs**

976 Figure 10 shows the kinds of association relationships that emerge when the PDRs are used in
 977 combination. The Numeric Sensor PDR in this example has Entity Identification Information that
 978 corresponds to "Power Supply 2." The containerID information in that Numeric Sensor PDR corresponds
 979 to the containerID that is linked to Physical Chassis 1 through the Entity Association PDR. Note that
 980 Physical Chassis 1 is identified as being contained only by the overall system. Hence, its containerID is
 981 SYSTEM.

982 Putting this information together yields a view of the system that is represented by the block diagram
 983 shown in Figure 10, which shows that the system contains a physical chassis that in turn contains two
 984 physical power supplies, and that each physical power supply has a temperature sensor associated with
 985 it. The link between the Numeric Sensor PDR and the entity it monitors/affects is [entityType,
 986 entityInstance, containerID]. See clause 10.3 Sensor/Effecter to Entity associations for definition and
 987 usage.



988

989

Figure 10 – Entity identification relationship between PDRs

990 The Entity Identification Information can thus be used for different types of associations within the PDRs.
 991 In this example, it is used in the Numeric Sensor PDR to identify the monitored entity in a sensor-to-entity
 992 association, and it is used within an Entity Association PDR to identify a containment association between
 993 the power supplies and the chassis.

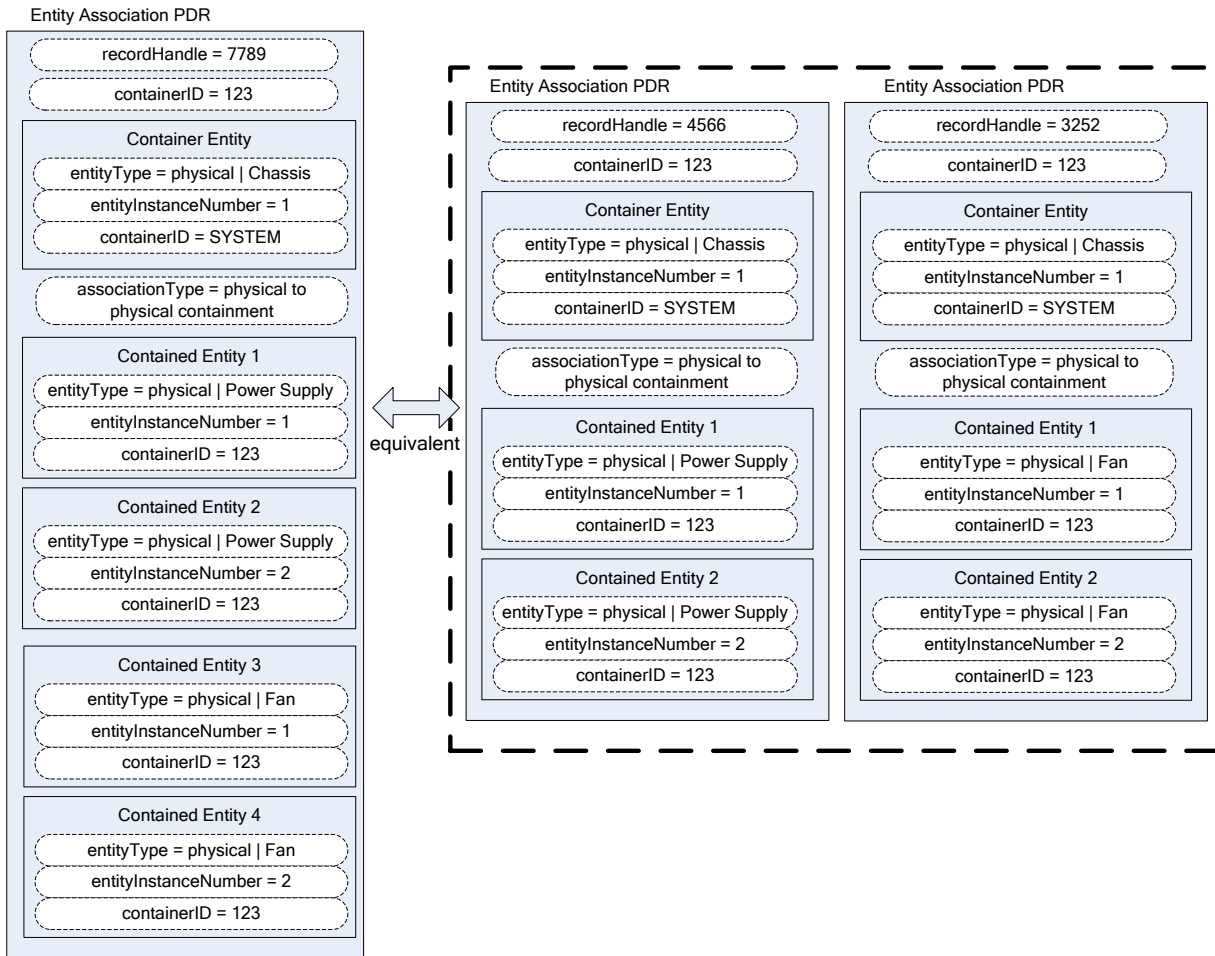
994 **11.3 Linked Entity Association PDRs**

995 Certain types of PDRs can be linked together using an Internal Association to form the equivalent of a
 996 single joint PDR. In Figure 11, the two Entity Association PDRs on the right are implicitly linked together
 997 by sharing the same containerID value. (Note that in Figure 11, the linked PDRs are also required to have
 998 the same container entity information and associationType values.)

999 The two PDRs on the right and the large single PDR on the left represent exactly the same association
 1000 relationship: the container entity "physical chassis 1" contains two physical power supplies, "power supply
 1001 1" and "power supply 2", and two physical fans, "fan 1" and "fan 2".

1002 It is a choice of the implementation whether a single PDR or multiple PDRs are used to represent a
 1003 containment association. Some implementations might want to use multiple records to make it easier to

1004 develop and maintain the records. For example, if a new physical entity is added for the chassis, it might
 1005 be more convenient to create a new PDR and link it into the existing containment PDRs for a chassis
 1006 rather than extending an existing containment PDR.



1007

1008

Figure 11 – Linked Entity Association PDRs

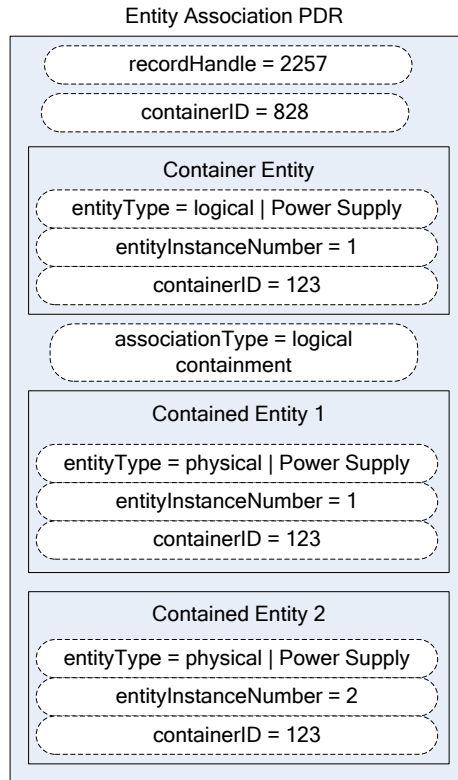
1009 **11.4 Logical containment associations**

1010 Entity Association PDRs can also be used to represent the relationship between logical entities and other
 1011 entities. A logical containment association identifies which physical and logical entities are contained in a
 1012 given logical container entity. A logical containment association can also consist of a physical container
 1013 entity that contains logical entities.

1014 This type of association is typically used to group items that have a common parameter that is monitored
 1015 or controlled. For example, power supplies might be grouped into a logical power supply because they
 1016 form a redundant power supply subsystem.

1017 The example PDR in Figure 12 shows a logical power supply 1 that contains physical power supply 1 and
 1018 a physical power supply 2. In this example, the containerIDs in the enclosed Entity Identification
 1019 Information do not reference the containerID of this overall PDR, but instead reference a container entity
 1020 from a different PDR. This follows from the previous example where containerID 123 corresponds to
 1021 physical chassis 1. The explanation for this is provided in 11.5.

1022 A logical containment association can have logical entities, physical entities, or both as contained entities.
 1023 The container entity must always be defined as a logical entity.



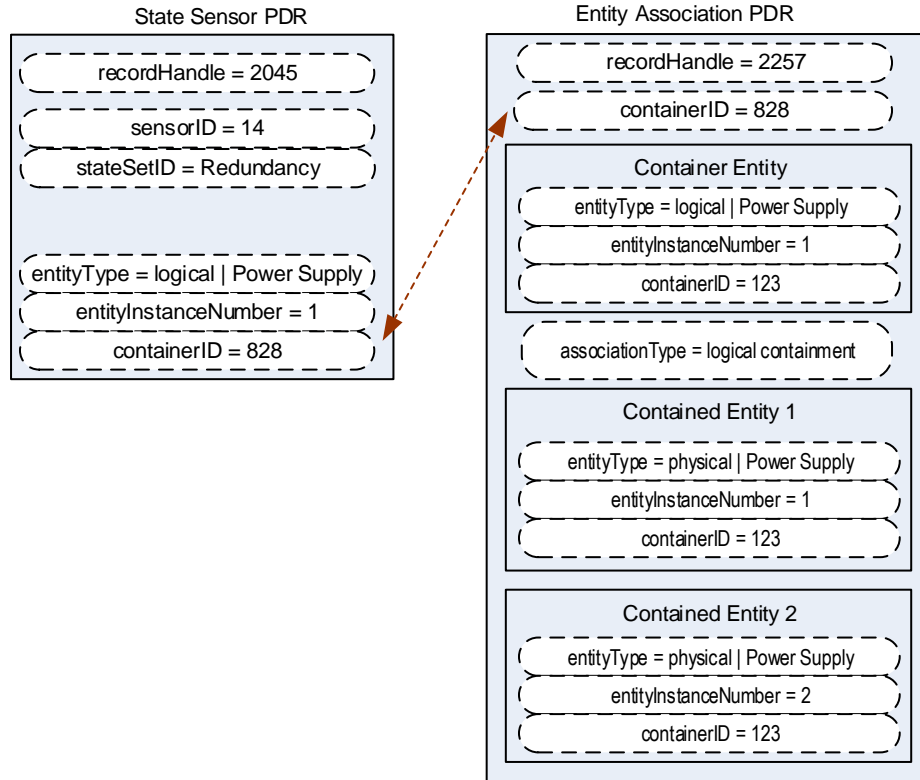
1024

1025

Figure 12 – Logical Containment PDR

1026 **11.5 Sensor/effector associations with logical entities**

1027 Sensors and effectors can be associated with logical entities in the same way that they can be associated
 1028 with physical entities. Figure 13 shows a state sensor that provides redundancy status and that has a
 1029 sensor-to-entity association to logical power supply 1. Note that containerID 123 follows from the previous
 1030 example where containerID 123 corresponds to physical chassis 1.



1031

1032

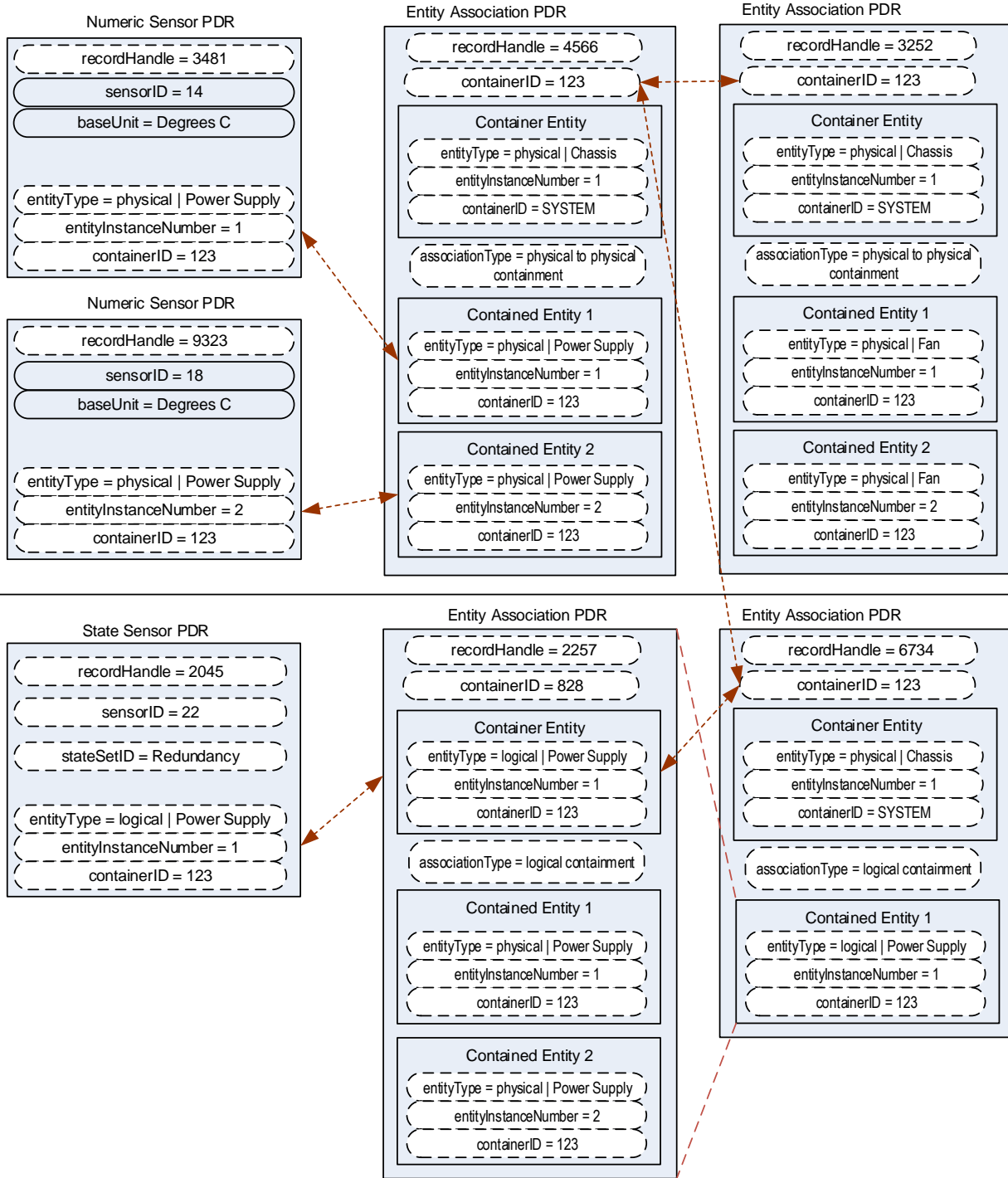
Figure 13 – Sensor/effector to logical entity association

1033 **11.6 Merged entity associations**

1034 Figure 14 presents a merged example that illustrates the different aspects and types of entity
 1035 associations that were introduced in previous subclauses 11.1 through 11.5. The PDRs in the top portion
 1036 of Figure 14 represent sensors and physical-to-physical containment associations. The lower half of
 1037 Figure 14 has PDRs that are related to the sensor and containment associations that define a logical
 1038 power supply. Together, these PDRs model a system that is represented in the block diagram shown in
 1039 Figure 15.

1040 The Entity Association PDR that defines the contained entities for logical power supply 1 uses 123 as the
 1041 containerID in the Entity Identification Information for the contained physical power supplies rather than
 1042 828, the containerID for the logical association, for the following reasons:

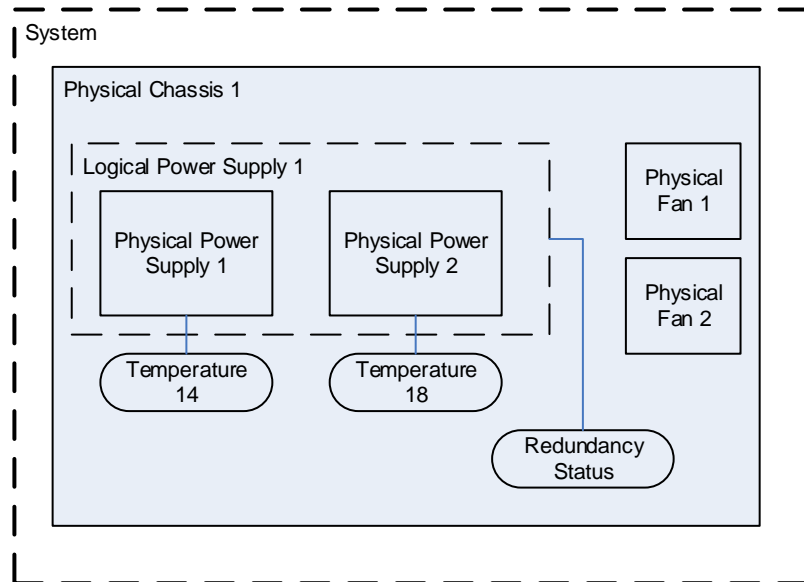
- 1043 • An entity that is contained in both physical and logical containment associations should use the
 1044 containerID that corresponds to a physical containment association.
- 1045 • The Entity Identification Information values for a given entity must be the same for all references
 1046 to the entity within the PDRs. A given entity cannot be identified using different container IDs in
 1047 different associations.



1048

1049

Figure 14 – Merged entity association PDR example



1050

1051

Figure 15 – Block diagram for merged entity association PDR example

1052

11.7 Separation of logical and physical associations

1053
1054
1055
1056
1057

Logical associations may be thought of as something that is layered on top of the physical association hierarchy. The previous example identifies container entity 123 (which corresponds to Physical Chassis 1) as the container entity for both physical and logical association PDRs. The types of associations are handled through separate PDRs, which separates the types of associations and helps avoid confusion when a given entity is part of more than one association.

1058
1059

Figure 15 highlights this by showing the physical-to-physical association PDRs in the upper part of the figure and the logical containment PDRs in the lower part.

1060

11.8 Designing association PDRs for monitoring and control

1061

Following is one method for creating or designing PDRs for a simple system:

1062
1063
1064
1065
1066
1067
1068
1069
1070
1071

- 1) Identify the physical entities and assign them Entity Identification Information values:
 - a) Identify the topmost physical container entities and give them the containerID for "system".
 - b) Assign each remaining physical entity a different containerID value using whatever approach works best for the implementation. (For example, containerID values could be assigned sequentially starting from 1, or 1000 if it necessary to have a value that is more readily distinguishable as a being a containerID.)
- 2) Create Entity Association PDRs for the physical-to-physical containment associations.
- 3) Create the Sensor PDR, Effector PDR, or other PDRs that are associated with the physical entities, and set the Entity Identification Information based on the containment PDRs that were created earlier.

- 1072 4) Create the PDRs for any logical entities and set the containerID value for the containing entity to
1073 the containerID for the appropriate physical container entities.
- 1074 5) Create the Sensor PDR, Effector PDR, or other PDRs that reference those logical entities.

1075 11.9 Terminus associations

1076 Many PDRs that are related to monitoring and control include a value called the PLDM Terminus Handle.
1077 This is an opaque value that is used solely within the PDRs in a given repository as a means of identifying
1078 the records that are associated with a particular terminus. The Terminus ID (TID) is a value that is used
1079 with PLDM messaging as a way to identify a particular terminus. A PDR called the PLDM Terminus
1080 Locator PDR is used to bind the PLDM Terminus Handle and the TID for a given terminus.

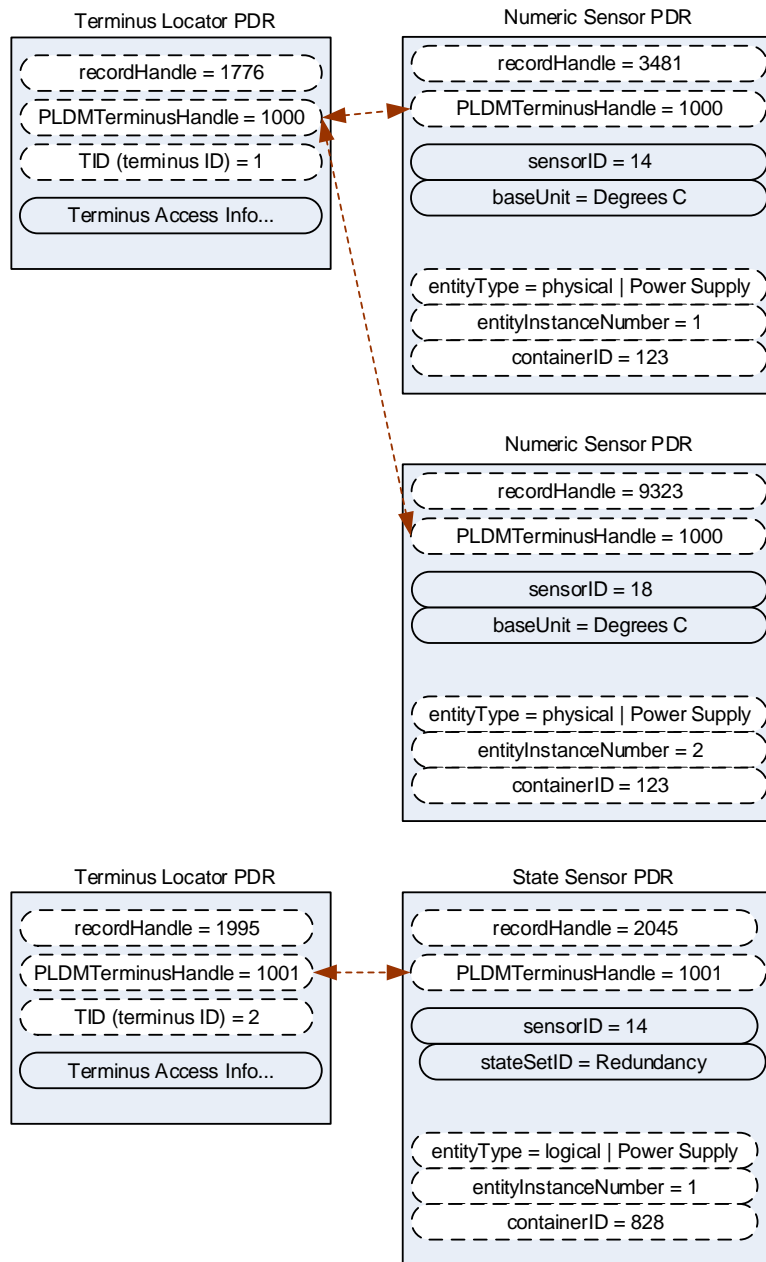
1081 An overview of PLDM Terminus Handles and TIDs is given in 12.1. Figure 16 provides an illustration of
1082 the relationship of the PLDM Terminus Handle and TID and how they are used within the PDRs.

1083 The association of entities with sensors and effectors is independent of the terminus that provides access
1084 to the sensor or effector. Sensors and effectors are associated with the entity that is being monitored or
1085 controlled rather than the entity that is providing the PLDM terminus that is used to access the sensor or
1086 effector. For example, if a system board entity has a voltage sensor and a temperature sensor, the
1087 voltage sensor could be provided through one terminus and the temperature sensor through a different
1088 terminus. Both sensors would be associated with the same system board entity, however.

1089 Because Entity Association PDRs may have content in them that has associations with more than one
1090 terminus, the PLDM Terminus Handle is used to identify which terminus *provided* the PDR rather than
1091 which terminus *is associated with* the PDR. For example, this information can be used to identify when
1092 PDR information has been provided by an add-in card so that the PDRs can be updated if the add-in card
1093 is removed. In many applications, such as mapping PLDM to CIM, the PLDM Terminus Handle
1094 information in an Entity Association PDR can be ignored.

1095 Figure 16 also shows how the PLDMTerminusHandle field is used to identify which sensor PDRs are
1096 accessed through a particular terminus. The example shows two different termini providing sensors for
1097 the system. The terminus with TID 1 is bound to PLDMTerminusHandle 1000 using the Terminus Locator
1098 PDR with recordHandle 1776; the terminus with TID 2 is bound to PLDMTerminus Handle 1001 using the
1099 Terminus Locator PDR with recordHandle 1995.

1100 PLDMTerminusHandle 1000 is associated with the PDRs for two numeric temperature sensors that are
1101 then associated with physical power supplies 1 and 2. PLDMTerminusHandle 1001 is associated with a
1102 single redundancy state sensor that is associated with logical power supply 1. Figure 17 shows a block
1103 diagram of these relationships. Note that while this example shows different termini monitoring different
1104 entities, different termini can also provide sensors that monitor a common entity. For example, one
1105 terminus could provide voltage sensors for a processor while another terminus could provide a
1106 temperature sensor for the same processor.



1107

1108

Figure 16 – TID and PLDM Terminus Handle associations

1109

Figure 17 shows a block diagram representation of a hypothetical system that is consistent with the terminus-to-sensor associations shown in Figure 16.

1110

1111

The example contains three management controllers. Management Controller 3 implements a PLDM terminus that includes a PLDM State Sensor that provides the redundancy status of logical power supply 1. Management Controller 2 implements a PLDM terminus that supports PLDM access to temperature sensors for physical power supplies 1 and 2. Management Controller 2 also holds the Primary PDR Repository for the system. Management Controller 1 represents a management controller or some other party that is accessing the PLDM subsystem. Management Controller 1 gets its view of the PLDM

1112

1113

1114

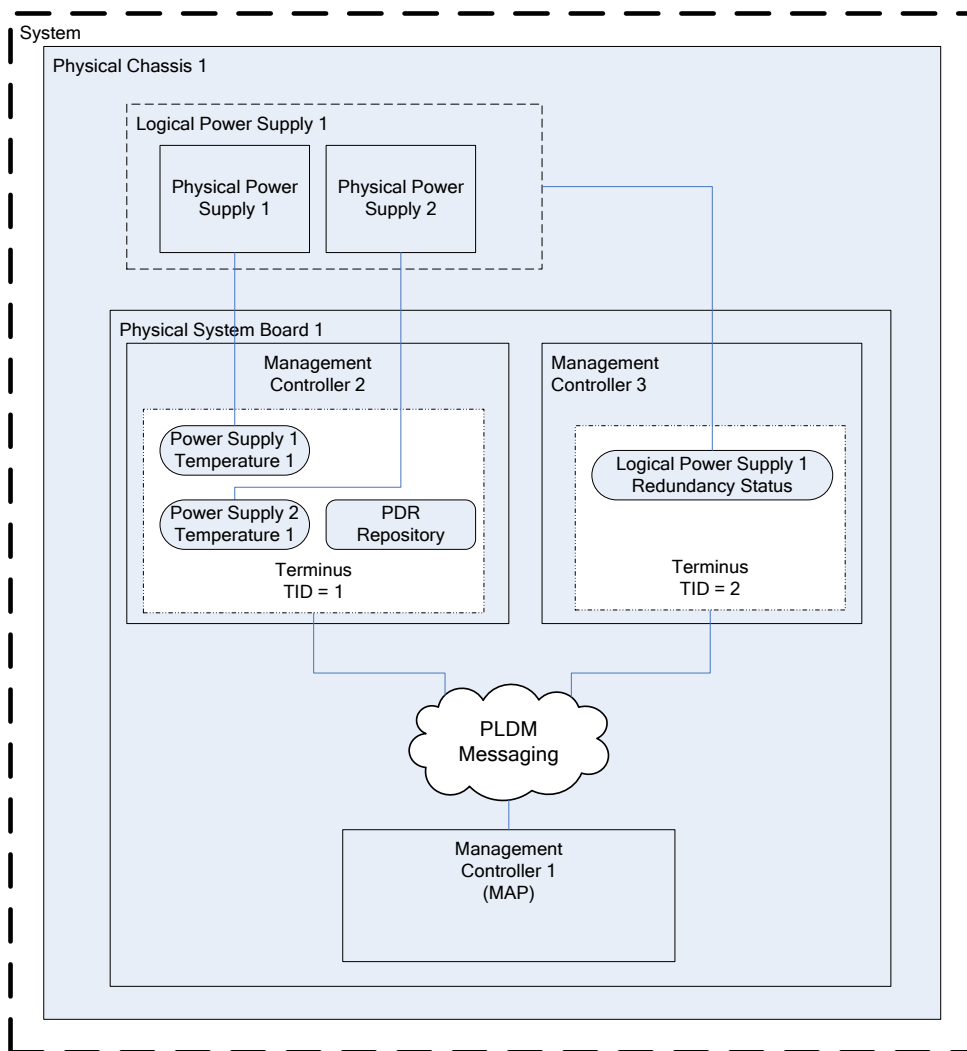
1115

1116

1117 subsystem by accessing the PDRs in the Primary PDR Repository provided by Management Controller 2.
 1118 Although this example shows one terminus per management controller, more than one terminus can be
 1119 implemented in a management controller.

1120 The PLDM Messaging cloud represents PLDM messaging connectivity between these three controllers.
 1121 In an actual implementation, this connectivity would be accomplished using a transport protocol and
 1122 physical medium that supports PLDM messaging, such as MCTP over SMBus/I²C.

1123 The example PDRs in Figure 16 are a subset of the PDRs that would be needed to represent the system
 1124 shown in Figure 17. For example, in addition to the Terminus Locator and Sensor PDRs, Entity
 1125 Association PDRs would identify that physical chassis 1 contains physical power supplies 1 and 2, logical
 1126 power supply 1, and a physical system board 1; that system board 1 contains Management Controllers 1,
 1127 2, and 3; and so on.



1128

1129

Figure 17 – Block diagram of Terminus-to-Sensor associations

1130 11.10 Interrupt associations

1131 Platform interrupts represent logical or physical signals that may be monitored or controlled by PLDM,
1132 such as NMI, IRQs, software interrupts, and so on. PLDM State Sensors and PLDM State Effecters can
1133 be used to monitor or control platform interrupts.

1134 11.10.1 Interrupt Association PDR

1135 PLDM includes a type of Association PDR called an Interrupt Association PDR that can be used to
1136 identify the relationship between one or more interrupt source entities and the target entity for a platform
1137 interrupt. The Interrupt Association PDR also identifies which sensor or effector is associated with the
1138 source entity. (Because a given target may receive interrupts from multiple sources, the sensor or effector
1139 is typically associated with the source entity rather than the target entity.)

1140 Two kinds of interrupts can be monitored by a state sensor:

- 1141 • **Received** interrupt associations identify when an interrupt target entity has received an interrupt
1142 from an interrupt source entity.
- 1143 • **Requested** interrupt associations identify when an interrupt source has issued an interrupt
1144 request to an interrupt target entity.

1145 Received interrupts and requested interrupts have different state sets. Thus, received and requested
1146 interrupts are differentiated by the state set that is used with the sensor. Effecters will typically use only
1147 the state sets for requested interrupts.

1148 11.10.2 Interrupt Association example

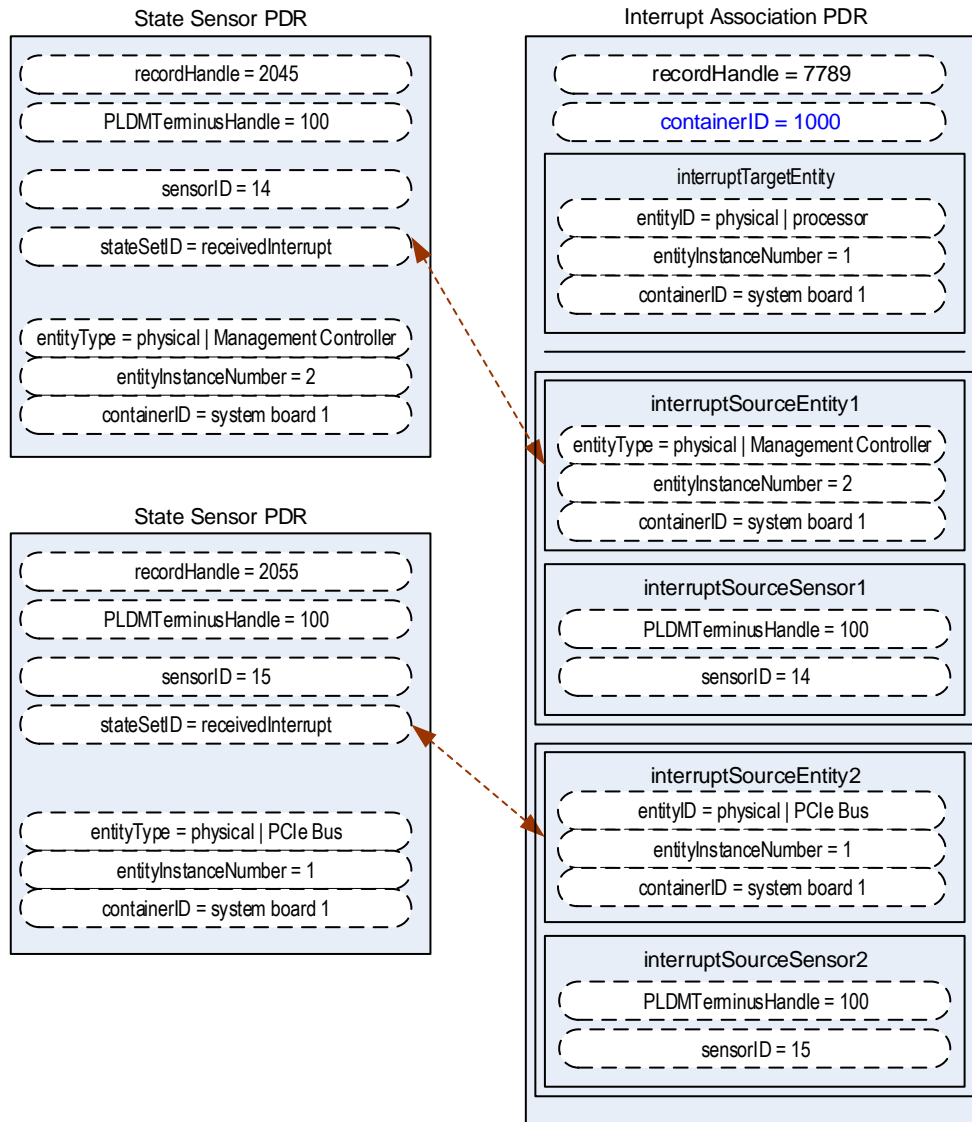
1149 This clause presents an example of using an Interrupt Association PDR. In this example, processor 1 is
1150 the interrupt target entity that is associated with PCIe Bus 1 and Management Controller 2 as potential
1151 interrupt source entities. Management Controller 1 provides the implementation of two sensors that report
1152 whether interrupts have been received from those sources.

1153 For this example, assume that each state sensor detected that an interrupt occurred and subsequently
1154 generated an event message on that state change. The event message itself indicates only that "Sensor
1155 14 in TID 2 has entered state x". The PDRs are used to interpret this information as follows:

- 1156 1) The TID that is received in the event message is used to locate the PLDM Terminus Locator
1157 record for the terminus. From this, the PLDMTerminusHandle is obtained.
- 1158 2) The PLDMTerminusHandle and sensorID value are used to locate the State Sensor PDR for the
1159 sensor that triggered the event message. This PDR indicates that the stateSetID equals the
1160 "Interrupt" state set. The state set definition indicates that the value "x" means "received
1161 interrupt detected".
- 1162 3) The Entity Identification Information in the State Sensor PDR indicates that the interrupt is
1163 associated with Management Controller 1, which implies that Management Controller 1 is the
1164 source entity for the interrupt.
- 1165 4) At this point, the combination of the information in the event message and the state sensor PDR
1166 yields the following interpretation of the event message:
 - 1167 – "Sensor 14 in TID 2 has detected that an interrupt has been received from Management
1168 Controller 1".
- 1169 5) This information does not identify the target of the interrupt, however. To identify the target, the
1170 PLDMTerminusHandle and sensorID are used to locate the Interrupt Association PDR that
1171 identifies the target.

1172 The format of the Interrupt Association PDR in Figure 18 is similar to that of the containment association
1173 PDRs shown earlier. The main difference is that sensorID information is provided in conjunction with the

1174 Entity Identification Information for the interrupt source entities. This additional information is required
 1175 because a given source entity may be the source of more than one interrupt. The sensorID information
 1176 provides the mechanism for differentiating different interrupts from the same interrupt source entity.



1177

1178 **Figure 18 – Received interrupt association example**

1179 **12 PLDM terminus**

1180 A PLDM terminus is the point of communication termination for PLDM messages and the PLDM functions
 1181 associated with those messages. A terminus must be uniquely identifiable so that PLDM PDRs can
 1182 associate semantic information with it. Additionally, a terminus must be identifiable when it generates

1183 asynchronous messages, such as event messages. This identification is accomplished through a value
1184 called the Terminus ID (TID).

1185 **12.1 TIDs, PLDM Terminus Handles, and Terminus Locator PDRs**

1186 The TID is primarily used in PLDM messages to identify which terminus generated an asynchronous
1187 message, such as an event message. The PLDM Terminus Handle is a value that is used within a PDR
1188 Repository to identify PDRs that are associated with a particular terminus. Thus, the PLDM Terminus
1189 Handle is defined only within the scope of a particular PDR Repository. A PDR called the Terminus
1190 Locator PDR is used to associate a TID with a Terminus Handle. The Terminus Locator PDR also
1191 includes information that describes how the terminus is accessed using PLDM messaging.

1192 **12.2 Requirements for unique TIDs**

1193 The assignment of unique TIDs to termini is required in the following situations:

- 1194 • Unique TIDs are required for implementations that use PDRs for describing sensors, effecters,
1195 and associations within and among termini.
- 1196 • Unique TIDs are required when an implementation exposes a PLDM Event Log in order to
1197 discriminate events from different termini when reading the log.

1198 **12.3 Terminus messaging requirements**

1199 PLDM termini that meet this specification must implement PLDM Request (command) and Response
1200 messages per [DSP0240](#). Additionally, a Management Controller that implements the Event Receiver
1201 function must be able to accept and process at least one Event Message request while it is processing
1202 other (non-Event Message) requests. Similarly, a device that generates Event Messages must be able to
1203 accept an incoming request while it is waiting for the response for the event message.

1204 It is recommended that a terminus can accept and track requests from multiple requesters if the terminus
1205 is used in an implementation where it is likely to receive simultaneous requests from multiple parties.

1206 **12.4 Terminus Locator PDRs**

1207 The Terminus Locator PDR forms the association between a TID and PLDM Terminus Handle for a
1208 terminus. The Terminus Locator PDR thus binds a given terminus and the semantic information that is
1209 provided through the PDRs for the terminus. Figure 19 illustrates the relationship between a TID and
1210 PLDM Terminus Handle.

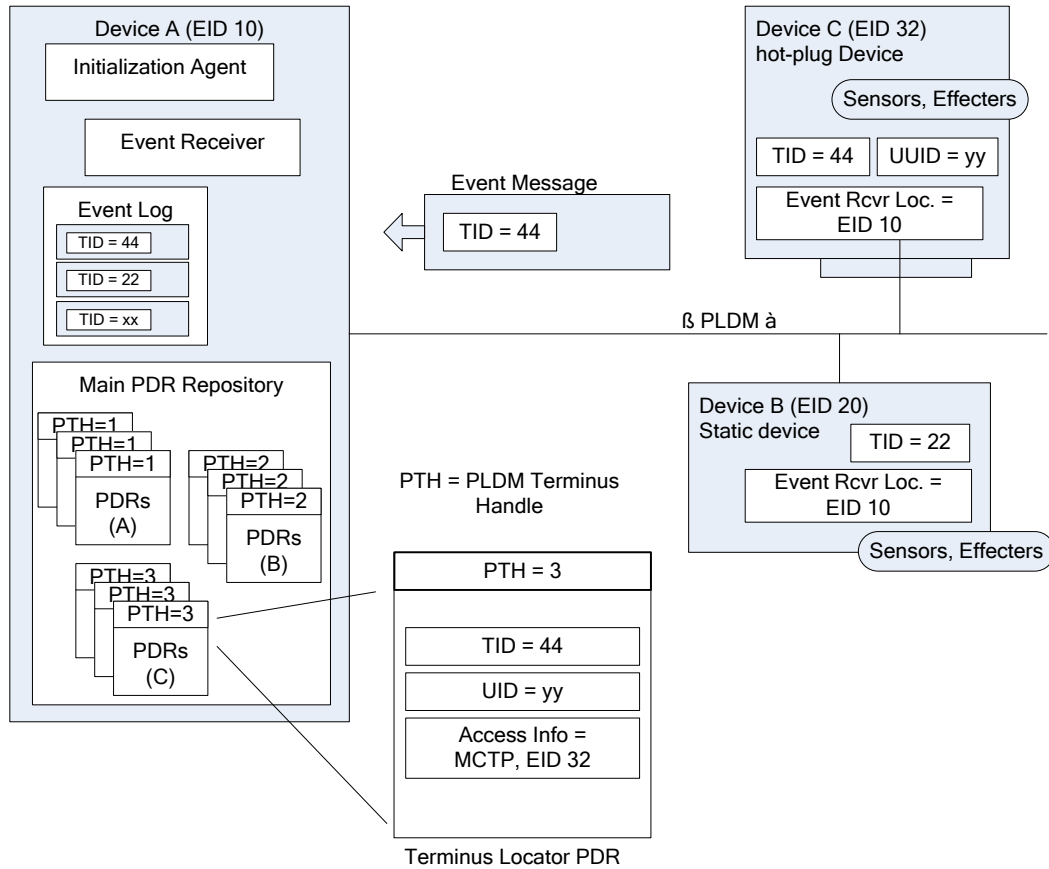
1211 The Terminus Locator PDR also provides additional information about a terminus, such as how it can be
1212 accessed through PLDM messages (hence the name "Terminus Locator"), and whether the terminus and
1213 set of PDRs associated with that terminus should be considered present.

1214 If the terminus has a UID or UUID, the Terminus Locator PDR may also hold a copy of the UID/UUID
1215 value. This value provides an additional mechanism to help verify that the PDRs associated with the
1216 terminus are correct for the particular terminus instance.

1217 The relationship between the PDRs and PLDM Messaging to and from a given terminus is identified using
1218 the following data in the Terminus Locator PDR. (This information is expressed using multiple fields within
1219 the actual record format.)

- 1220 • The PLDM Terminus Handle is used to identify PDRs that are associated to a particular
1221 terminus. It is used only within the scope of a particular PDR Repository.
- 1222 • The TID identifies a terminus for PLDM messaging, particularly for identifying messages that
1223 come from a given terminus. A PLDM Terminus Locator PDR associates the TID with the PLDM
1224 Terminus Handle that is used for accessing the PDRs that are associated with the terminus.

- 1225 • The Terminus Access Info consists of a list of protocols and additional information, such as
1226 addressing, which enables a party to send PLDM messages to the terminus.



1227

1228 **Figure 19 – Example of TID and PLDM Terminus Handle relationships**

1229 **12.5 Enumerating termini**

1230 A party that accesses the Primary PDR Repository can use the PDRs to enumerate the termini by listing
1231 and examining the Terminus Locator PDRs.

1232 **12.5.1 General**

1233 To support alternative platform configurations and hot-plug devices, the PDR Repository may have PDRs
1234 in it for termini that might not be present. This enables the PDR Repository to hold a superset of
1235 information for the possible termini that might be installed in the system. This helps enable
1236 implementations that support different configurations of termini using a preconfigured, static set of PDRs.

1237 To support this, the Terminus Locator PDR contains a field that indicates whether the record itself is valid.
1238 A terminus may also have a state sensor associated with it that reports whether the terminus is present
1239 and available for use (described in 12.5.3).

1240 The following rules apply to using Terminus Locator PDRs for enumerating termini. When it is stated that
1241 a terminus should be ignored, it is not an error condition. It means that the status of the terminus is
1242 unknown and from a PLDM point-of-view should be treated as if it did not exist at all.

1243 • A terminus must have a Terminus Locator PDR that is marked as valid in order to be
1244 considered present. Only one Terminus Locator PDR is allowed to be valid at a time for a given
1245 PLDM Terminus Handle within a PDR Repository. It is an error condition if multiple Terminus
1246 Locator PDRs exist and are simultaneously marked as valid for a given PLDM Terminus
1247 Handle.

1248 • If the terminus has a sensor associated with it that reports Terminus State, the sensor must
1249 indicate that the terminus is present. Otherwise, the terminus and its associated PDRs should
1250 be ignored.

1251 • If the terminus has a sensor associated with it that reports Terminus State and the Terminus
1252 State information cannot be accessed because the operationalState of the sensor is not
1253 "enabled", the terminus and its associated PDRs should be ignored.

1254 12.5.2 Unlisted or absent termini

1255 PDRs for a particular terminus should be ignored under the following conditions:

- 1256 • The PDR does not have an associated Terminus Locator PDR.
1257 • The PDR is related to a terminus that has an associated Terminus Locator PDR that is marked
1258 invalid or is not present based on a presence sensor.

1259 References to termini (for example, PLDM Terminus Handles) should be ignored under the following
1260 conditions:

- 1261 • The reference does not have an associated Terminus Locator PDR.
1262 • The reference is associated with a Terminus Locator PDR that is marked invalid or is not
1263 present based on a presence sensor.

1264 These conditions do not apply to OEM or vendor-defined PDRs.

1265 12.5.3 Terminus presence using Terminus State Sensors

1266 In some implementations, termini may need to be added or removed as devices are added to or removed
1267 from the platform or as platform configurations are changed. This can be handled by updating the validity
1268 field in the Terminus Locator PDRs or by updating the PDRs to add or remove Terminus Locator PDRs.
1269 Correspondingly, other PDRs that are associated with the terminus may also be updated, added, or
1270 removed. Updating PDRs may not be warranted in some implementations, such as when the
1271 implementation would have otherwise been able to use a static configuration of PDRs.

1272 A more dynamic way of indicating terminus presence is to associate a terminus with a "Terminus State
1273 Sensor". A Terminus State Sensor is a type of PLDM Composite State Sensor that is associated with a
1274 logical entity of type "PLDM Terminus" using a sensor to entity association. The sensor returns state set
1275 enumerations for "Presence status" and "Operational status". A Terminus State Sensor may be
1276 implemented as a sensor at the terminus itself, or it may be implemented as a sensor under another
1277 terminus.

1278 13 PLDM events

1279 PLDM events are primarily related to changes of PLDM sensor states or states that are related to the
1280 operation of PLDM or the PLDM subsystem itself.

1281 NOTE PLDM events are not the same as CIM indications. There will typically not be a one-to-one correspondence
1282 between PLDM events and CIM indications. In some cases, a PLDM event may trigger a MAP to generate

1283 indications or entries in a CIM record log, while in other cases a PLDM event may be used solely to update
1284 CIM properties to eliminate or reduce polling by the MAP, or to report information about the internal health or
1285 operation of the PLDM subsystem that is not exposed through CIM.

1286 PLDM Events are between a PLDM terminus and the PLDM Event Receiver (such as a management
1287 controller). PLDM Events may be shared externally using the PLDM Event Log. The method to share the
1288 PLDM Event Log is outside the scope of this specification.

1289 **13.1 PLDM Event Messages**

1290 PLDM Event Messages are PLDM monitoring and control messages that are used by a PLDM terminus to
1291 synchronously or asynchronously report PLDM events to a central party called the PLDM Event Receiver.
1292 This specification version also adds a method to allow the event receiver to poll for events from the PLDM
1293 terminus event log.

1294 The PlatformEventMessage command supports multiple Event Data Classes.

1295 The PLDM terminus is expected to maintain an internal event message FIFO (queue) for both
1296 asynchronous transmission and polled message requests; All PLDM Event Messages are acknowledged
1297 by the PLDM Event Receiver using the command specific method. The number of entries in the PLDM
1298 terminus FIFO (queue) is implementation specific but should be sufficient to hold early events that occur
1299 before the PLDM Event Receiver configures the PLDM terminus for events. The FIFO should allow at
1300 least one event entry for each enabled sensor.

1301 The PLDM Event Receiver can only poll or accept PLDM Event Messages from the terminus after the
1302 terminus responds to the 16.4 SetEventReceiver command. The PLDM terminus may overwrite the oldest
1303 event (entry) or the oldest event for a specific sensor entry in the FIFO when the terminus (event) queue
1304 is full. Once a terminus transmits an event, the PLDM Event Receiver must acknowledge the event using
1305 the command specific acknowledgement. The acknowledged events are removed from the FIFO.

1306 There are two methods to transmit an event message to the event receiver:

1307 1. 16.6 PlatformEventMessage command

1308 This command allows the PLDM terminus to asynchronously transmit a PLDM event message to
1309 the established and designated PLDM Event Receiver. The Event Receiver acknowledges
1310 receiving the PLDM Event Message in the response to this command. DSP0240 (PLDM Base
1311 Specification) provides timing parameters in "Table 5 – Timing Specifications for PLDM
1312 Messages". The PLDM terminus is the Requester and shall retry sending this command "Number
1313 of request retries" (DSP0240, Table 5).

1314 2. 16.7 PollForPlatformEventMessage

1315 This command allows the designated PLDM Event Receiver to synchronously request (poll for) a
1316 PLDM terminus event message. The PLDM Event Receiver retrieves a single PLDM event
1317 message on each poll and should poll the terminus until the terminus indicates no more events.
1318 After the initial request (poll), the PLDM Event Receiver shall acknowledge the event returned on
1319 the next request (poll). The terminus may remove the event from the FIFO when the
1320 acknowledgement is received.

1321 **13.2 PLDM Event Receiver**

1322 The destination for event messages within PLDM is called the Event Receiver. The Event Receiver
1323 function is implemented by a PLDM terminus within the platform management subsystem. Multiple termini
1324 can send Event Messages to the Event Receiver function. The SetEventReceiver command is used to
1325 give the location of the Event Receiver function to termini that generate event messages.

1326 A PLDM Subsystem is defined as the collection of devices enumerated by the same PLDM initialization
 1327 agent.

1328 A PLDM subsystem implementation can have only one PLDM Event Receiver function enabled at a given
 1329 time. It is expected that typical implementations will always assign the same Event Receiver location.
 1330 However, the location of the Event Receiver function is allowed to be changed during PLDM subsystem
 1331 operation. For example, some implementations may do this to support a failover of the Event Receiver
 1332 function, or to migrate it to a management controller that is hot plugged into the system, and so forth.

1333 **13.3 PLDM Event Logging**

1334 PLDM Event Logging defines an interface through which event messages that have been received at the
 1335 Event Receiver can be saved in an area of storage called the PLDM Event Log for later retrieval. Event
 1336 logging includes mechanisms for storing and time-stamping event records, determining characteristics of
 1337 the log (such as its capacity), and reading and clearing the contents of the log.

1338 Additionally, "virtual" PLDM Event Messages may be internally generated within the terminus that is
 1339 providing the PLDM Event Log function and directly logged without being received as PLDM Event
 1340 Messages on any external interface.

1341 A PLDM terminus shall be tied to at most one PLDM Event Receiver and at most one PLDM Event Log
 1342 function. The PLDM Event Log function is expected to be provided by a "time aware" management
 1343 controller for the PLDM Subsystem. A simple PLDM terminus supporting a device or adapter should
 1344 maintain an internal structure to support the 16.6 PlatformEventMessage command or the 16.7
 1345 PollForPlatformEventMessage . The definition of this internal structure is implementation specific and
 1346 outside the scope of this specification.

1347 Additional information about event logging is provided in clause 23.

1348 **13.4 PLDM Event Log clearing policies**

1349 The PLDM Event Log can use different policies for automatically clearing entries from the log (Table 5).
 1350 The active policy is configured through the SetPLDMEventLogPolicy command. Refer to the specification
 1351 of this command for policy support requirements.

1352 **Table 5 – PLDM Event Log clearing policies**

Policy	Description
Fill and Stop	The PLDM Event Log stops accepting new entries after it has become full. The log does not automatically clear. It must be cleared using the ClearPLDMEventLog command. This policy does not utilize any parameters.
FIFO	When the log is full, the oldest <i>N</i> entries are automatically deleted when the next entry is received. This policy uses a single parameter, <i>N</i> . <i>N</i> may be a fixed or configurable parameter, depending on the implementation. An implementation can also express <i>N</i> as a percentage of the log (<i>N</i> Percentage) instead of as an integral number of entries.

Policy	Description
Clear on Age	<p>When the log has filled past a threshold number of entries, M, the age of the first N entries is checked to see if they have been in the log for more than a given age interval. If the Mth entry is older than the age interval, the first N entries are automatically cleared from the log. If the log is less than M entries full, entries are retained indefinitely, regardless of their age.</p> <p>This policy uses three parameters: Age, N, and M. The Age interval, the number of automatically cleared entries, N, and the threshold value, M, may be fixed or configurable parameters, depending on the implementation. The policy may also be implemented with N and M given as percentages of the log (MPercentage and NPercentage) instead of an integral number of entries.</p>

1353 **13.5 Oldest and newest log entries**

1354 Unless otherwise specified, when the terms *old*, *older*, *oldest*, *new*, *newer*, and *newest* are used to refer
 1355 to PLDM Event Log entries, the terms refer to the time that the event was entered into the log rather than
 1356 the timestamp of the entry. This is because the setting of the log timestamp clock might be changed
 1357 during system operation, making it possible for temporally newer log entries to have timestamps that refer
 1358 to an older time than temporally older entries.

1359 **13.6 Event Receiver Location**

1360 The information that is used by a given terminus to send messages to the Event Receiver function (such
 1361 as addressing) is referred to as the Event Receiver Location information. Event Receiver Location
 1362 information is transport dependent; for example, for MCTP the information would consist of the EID
 1363 (MCTP Endpoint ID) of the Event Receiver. Additionally, the Event Receiver Location information may
 1364 vary on a per-terminus basis, depending on the requirements of the transport and medium. The PLDM
 1365 Transport binding specifications define how the Event Receiver Location is set for a particular transport
 1366 and medium.

1367 PLDM supports a SetEventReceiver command that enables the Event Receiver Location information to
 1368 be delivered to termini that generate event messages. This approach provides the following
 1369 characteristics:

- 1370 • It eliminates the need to specify a well-known address for the Event Receiver function for each
 1371 different medium and transport.
- 1372 • It supports assigning the Event Receiver function to a different location, which could be used to
 1373 – support failover of the Event Receiver function to another device
 1374 – enable the Event Receiver function to be handled by an alternative device that gets added
 1375 into the system
 1376 – support a situation in which the Event Receiver function is on a medium where its address
 1377 changes during PLDM operation
- 1378 • It provides a mechanism that helps synchronize the generation of event messages with the
 1379 availability of the Event Receiver function.
- 1380 • It provides a mechanism to allow synchronous (polling) and asynchronous event messages to
 1381 be communicated to the Event Receiver.

1382 **13.7 PLDM Event Log entry formats**

1383 Table 6 shows the general format that is used for all PLDM Event Log entries.

1384

Table 6 – PLDM Event Log entry format

Byte	Type	Field
0	enum8	entryType value: { PLDMPlatformEvent, OEMTimestampedEntry, OEMEntry }
1	uint8	entryDataLength The size in bytes of the entryData field.
variable	–	entryData Data for the entry, dependent on the entryType. If entryType = PLDMPlatformEvent, the entryData format is given in Table 7. If entryType = OEMTimestampedEntry, the entryData format is given in Table 8. If entryType = OEMEntry, the entryData format is given in Table 9.

1385 **13.8 PLDM Platform Event Entry Data format**

1386 Table 7 specifies the format used for the entryData field in PLDM Event Log entries that use the
1387 PLDMPlatformEvent value for the entryType field.

1388

Table 7 – Platform Event Entry Data format

Byte	Type	Field
0	sint8	entryTimestampUTCOffset The UTC offset for the log entry timestamp in increments of 1/2 hour special value: 0xFF = unspecified
1:5	uint40	entryTimestampSeconds This value corresponds to a 40-bit unsigned integer that represents the number of seconds since midnight UTC of January 1, 1970 (not counting leap seconds).
6	uint8	entryTimestamp100s This value provides a number of 1/100ths of a second added to entryTimestampSeconds. value: 0 to 99 special value: 0xFF = unspecified. Use this value if the implementation timestamps entries to no finer than a one-second resolution.
variable	–	eventData The eventData format is the same as the format for the request parameters of the PlatformEventMessage command (see Table 15).

1389 **13.9 OEM Timestamped Event Entry Data format**

1390 Table 8 specifies the format used for the entryData field in PLDM Event Log entries that use the
 1391 OEMTimestampedEntry value for the entryType field.

1392 **Table 8 – OEM Timestamped Event Entry Data format**

Byte	Type	Field
0:3	uint32	vendorIANA The IANA Enterprise Number for the vendor that is defining the OEMData. The list of Enterprise Numbers can be found at www.iana.org/protocols/ . special value: 0 = unspecified.
4	sint8	entryTimestampUTCOffset The UTC offset for the log entry timestamp in increments of 1/2 hour special value: 0xFF = unspecified
5	uint40	entryTimestampSeconds This value corresponds to a 40-bit unsigned integer that represents the number of seconds since midnight UTC of January 1, 1970 (not counting leap seconds).
10	uint8	entryTimestamp100s This value provides a number of 1/100ths of a second added to entryTimestampSeconds. value: 0 to 99 special value: 0xFF = unspecified. This value is used if the implementation timestamps entries to no finer than a one-second resolution.
variable	variable	OEMData OEM-specific data that is specified by the vendor identified by vendorIANA

1393 **13.10 OEM Event Entry Data format**

1394 Table 9 specifies the format used for the entryData field in PLDM Event Log entries that use the
 1395 OEMEntry value for the entryType field. The format is similar to the OEM Timestamped Event Entry Data
 1396 format (shown in Table 8), except that it does not include PLDM-defined timestamp fields.

1397 **Table 9 – OEM Event Entry Data format**

Byte	Type	Field
0:3	uint32	vendorIANA The IANA Enterprise Number for the vendor that is defining the OEMData special value: 0 = unspecified
variable	variable	OEMData OEM-specific data that is specified by the vendor identified by vendorIANA

1398 **14 Discovery Agent**

1399 The Discovery Agent function is responsible for discovering termini, assigning them unique TID values,
 1400 and assigning them the address of the Event Receiver function.

1401 If the implementation is maintaining a Primary PDR Repository, the Discovery Agent may also be required
1402 to automatically create or update PDRs to support devices such as hot-plug devices that may be
1403 dynamically added or removed from the system. This includes the following actions:

- 1404 • creating records such as Terminus Locator PDRs
- 1405 • extracting Device PDR information and merging it into the Primary PDR Repository
- 1406 • updating associating records to link Device PDR information into the overall context of the
1407 platform management subsystem

1408 Any OEM PDRs in the Device PDR information that are identified to be copied to the Primary PDR
1409 Repository are also added to the Primary PDR Repository by the Discovery Agent.

1410 **14.1 Assignment of TIDs and Event Receiver location**

1411 Following are the support requirements for assignment of TIDs and the launching of the Initialization
1412 Agent by a Discovery Agent within a PLDM implementation:

- 1413 • All termini must support the SetTID command.
- 1414 • All termini that generate PLDM Event Messages shall support the SetEventReceiver command.
1415 Termini that do not generate PLDM Event Messages are not required to support the
1416 SetEventReceiver command. Those termini, however, that support “Polled Events” shall support
1417 the SetEventReceiver command.
- 1418 • The Discovery Agent function is responsible for discovering termini and assigning them unique
1419 TID values. (A default TID setting may be preconfigured for a PLDM terminus if the terminus is
1420 statically configured into the platform. This setting must be able to be overridden using the
1421 SetTID command.)
- 1422 • The Initialization Agent function is responsible for initializing PLDM sensors and effecters and
1423 setting Event Receiver location information into the termini. (A default Event Receiver setting
1424 may be preconfigured for a PLDM terminus if the terminus is statically configured into the
1425 platform. This setting must be able to be overridden using the SetEventReceiver command.)
1426 The Initialization Agent function is described in more detail in clause 15.
- 1427 • When PDRs are used, the Initialization Agent is also responsible for maintaining corresponding
1428 Terminus Locator PDR information.
- 1429 • A terminus must have its Event Receiver information set before it can begin to issue PLDM
1430 Event Messages.
- 1431 • A terminus that has standby power should retain its TID and Event Receiver settings. When the
1432 terminus comes back online, it can use that information for event messaging without requiring
1433 Event Receiver reinitialization.
- 1434 • A terminus should retain its TID and Event Receiver settings during a given PLDM subsystem
1435 operation.
- 1436 • Termini that are to be rediscovered (that is, termini that are not statically configured into the
1437 system and may lose PLDM communication temporarily, which might occur in different platform
1438 power states) must have a separate unique and persistent ID that can be associated with the
1439 terminus. For example, if a terminus is hot-plug, it should have a universally unique ID (UUID).
- 1440 • TIDs are not required to persist or remain constant across PLDM subsystem restarts, unless the
1441 system is using PDRs or exposes a PLDM Event Log. In such cases, TIDs must be persistently
1442 stored by the termini or reassigned to the same value by the Discovery Agent function.
- 1443 • A MAP or other entity that is accessing a PLDM subsystem should not cache TIDs because
1444 TIDs might change if the PLDM subsystem is reset or reinitialized.

- 1445
- 1446
- Termini on hot-plug cards must have a UUID or be associated with a terminus on the same card that has a UUID.
- 1447
- Implementations that do not use PDRs can assign TIDs in any manner, including not assigning them at all. In this case, the implementation must define its own mechanisms for identifying and tracking termini and event messages from termini.
- 1448
- 1449

1450 **14.2 UUIDs for devices in hot-plug or add-in card applications**

1451 If the device is intended to be used on an add-in or hot-plug card, it may be required to support a
1452 universally unique ID (UUID) depending on higher-level system requirements or initiatives. In general,
1453 add-in cards that plug into standardized I/O connections and are used in multiple vendor systems, such
1454 as PCIe add-in cards, are required to use UUIDs so that multiple instances of the same card can be
1455 detected.

1456 **14.3 UID implementation**

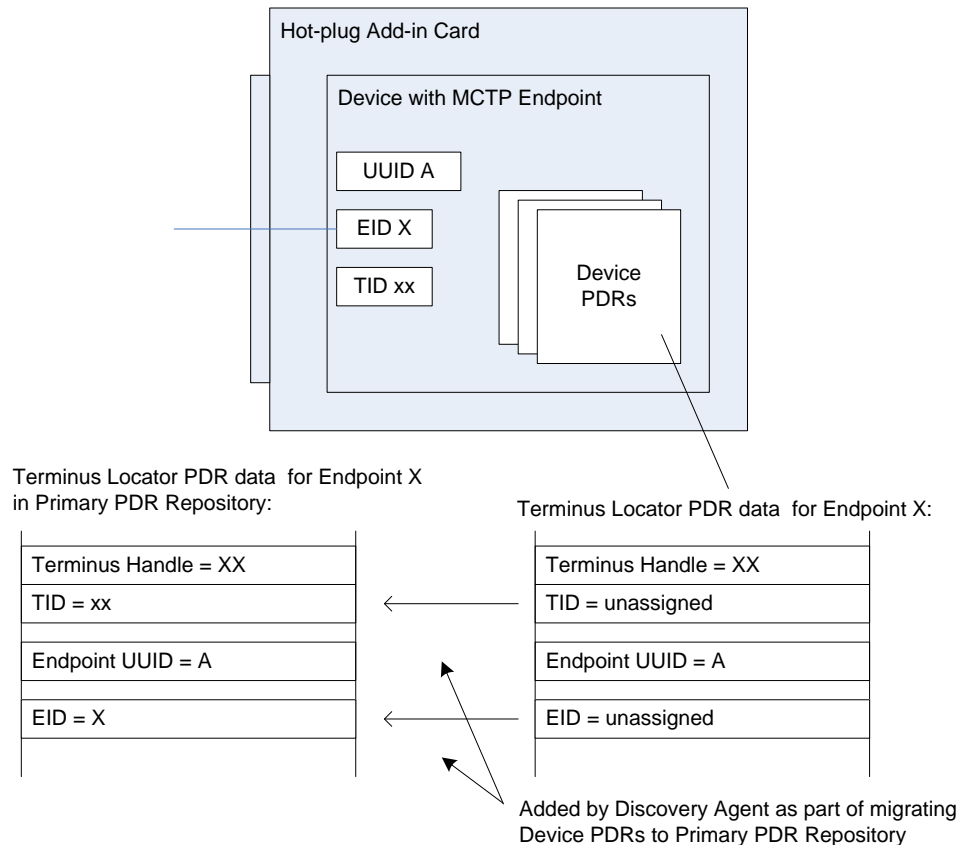
1457 If a terminus is required to have a unique ID (UID), how the UID is implemented depends on the
1458 component and how the device manufacturer intends the device to be used in a system. For example, it
1459 is the device manufacturer's choice whether the entire UID must be configured by the system integrator
1460 after purchasing the device, or a number of preconfigured UIDs in the device are selectable by a pin or
1461 nonvolatile configuration selection, or the UID is permanently embedded in the device. Typically, each
1462 device will have fuses, PROM, EPROM/EEPROM, or some other nonvolatile mechanism for holding the
1463 unique ID that is configured either during device manufacture or when the device is integrated into a
1464 system.

1465 **14.4 More than one terminus in a device**

1466 The Terminus Locator PDR contains a containerEntity field that can be used to identify the entity that
1467 contains the terminus. This field provides the mechanism to identify when multiple termini are within the
1468 same device or are located within the same entity.

1469 **14.5 Examples of PDR and UUID use with add-in cards**

1470 Figure 20 and Figure 21 present examples of how Device PDRs, UUIDs, and Terminus Locator PDRs
1471 work together to identify PLDM termini on add-in cards, such as hot-plug add-in cards, that may be
1472 dynamically inserted or removed during PLDM subsystem operation. Both examples illustrate MCTP-
1473 based implementations. However, the approach may be extrapolated to other transport types.



1474

1475

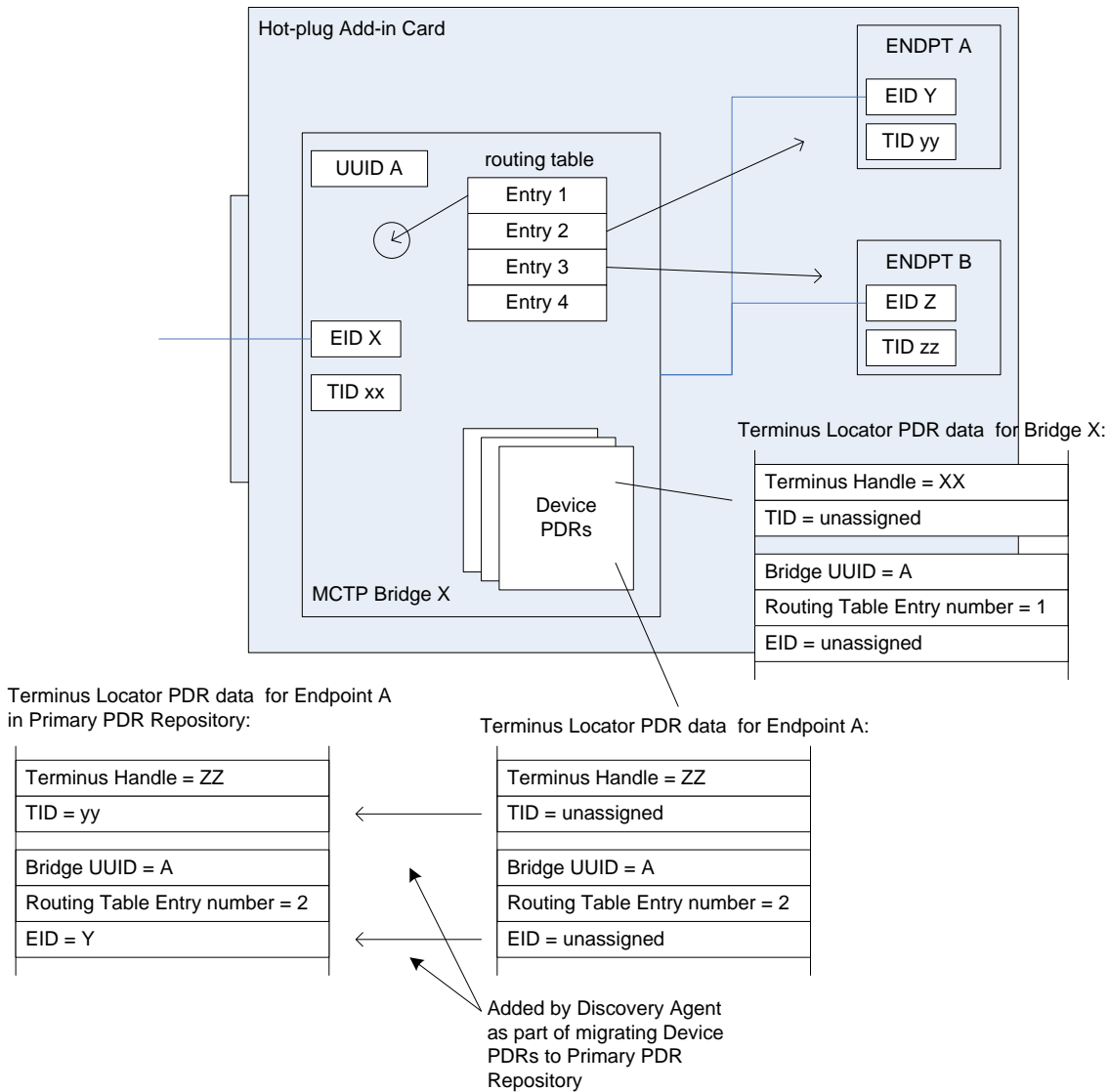
Figure 20 – Hot-plug add-in card with single PLDM terminus

1476 Figure 20 shows an add-in card that has a single PLDM terminus that is accessed through a single MCTP
 1477 endpoint. The terminus is persistently and uniquely identified within the PLDM subsystem by a UUID that
 1478 is associated with the endpoint and the terminus. This UUID is recorded in a partially filled-in Terminus
 1479 Locator PDR that is part of the Device PDRs that are provided by the add-in card. The UUID can also be
 1480 read by issuing a GetTerminusUID command to the terminus. The Device PDRs also report the presence
 1481 of and semantic information about sensors, effecters, and other functions on the add-in card.

1482 The Terminus Locator PDR from the Device PDRs returns "unassigned" values for the Endpoint ID (EID)
 1483 and Terminus ID (TID) fields because those values are unavailable before the card has been discovered
 1484 and initialized by MCTP and the PLDM Discovery Agent within the PLDM subsystem. It also eliminates
 1485 the need for the terminus to update those Device PDRs whenever TID or EID values are assigned or
 1486 changed. The Discovery Agent sets the TID for the terminus and adds the EID and TID values to the
 1487 Terminus Locator Record PDRs when they are integrated into the Primary PDR Repository. The
 1488 Discovery Agent then synthesizes other PDRs as necessary to link the add-in card into the overall
 1489 semantic information of the PLDM subsystem. For example, the Discovery Agent may create association
 1490 PDRs that associate the add-in card with a particular bus and connector within the system.

1491 The Discovery Agent is also responsible for keeping those records up-to-date if EID assignments change
 1492 during PLDM subsystem operation and for deleting or invalidating the PDRs that are associated with the
 1493 card and its termini if it detects that the card has been removed.

1494 Figure 21 shows an add-in card that has several MCTP endpoints, each with its own PLDM terminus.
 1495 One terminus is within an MCTP Bridge device that provides the Device PDRs for all the termini on the
 1496 card. Additionally, the MCTP Bridge provides a UUID that identifies the overall card for MCTP. All MCTP
 1497 endpoints are defined relative to MCTP Bridge function based on the position of their routing information
 1498 in the routing table.



1499

1500

Figure 21 – Hot-plug add-in card with multiple PLDM termini

1501 In Figure 21, the MCTP Bridge itself is associated with the first routing table entry, Endpoint A is
 1502 associated with the second entry, and Endpoint B is associated with the third entry. The Device PDRs
 1503 hold Terminus Locator PDRs for each terminus that is on the add-in card. These PDRs uniquely identify
 1504 each terminus using two pieces of information: the UUID of the MCTP Bridge and the position of a routing
 1505 table entry that is associated with the terminus. The routing table entry positions must not change during

1506 PLDM subsystem operation. This approach eliminates the need for Endpoints A and B to have their own
1507 support for UUIDs.

1508 **15 Initialization Agent**

1509 This clause describes the role and operation of the Initialization Agent function in a PLDM subsystem that
1510 uses PDRs.

1511 **15.1 General**

1512 PLDM sensors are not required to completely self-initialize and enable themselves upon PLDM
1513 subsystem startup or upon power state changes of the device that is hosting the sensor. Thus, low-cost
1514 devices are not required to have nonvolatile configuration resources. Additionally, the mechanism
1515 provides options for overriding default configurations of sensors and event generation.

1516 The Initialization Agent is a function that initializes message generation and sensor configuration as
1517 described by Sensor Initialization PDRs. The Initialization Agent function normally runs whenever the
1518 platform management subsystem is first powered up, upon system Hard and Soft Resets, and on certain
1519 other transitions. Fields in the Sensor Initialization PDRs indicate the system transitions on which a given
1520 sensor is initialized.

1521 The Initialization Agent is also responsible for setting the Event Receiver Location information and
1522 enabling event message generation.

1523 The Sensor Initialization PDRs hold information that describes the default threshold values, states, and
1524 event generation settings for sensors that are initialized by the Initialization Agent function. Sensor
1525 Initialization PDRs are required only for sensors that are initialized by the Initialization Agent. Sensors that
1526 are self-initializing or are initialized through some mechanism that is outside the PLDM specifications do
1527 not need Sensor Initialization PDRs.

1528 The Initialization Agent function thus eliminates the need for all sensors to retain their own nonvolatile
1529 storage for their default settings, and also provides a mechanism to retrigger any events that may have
1530 been transmitted before the Event Receiver function was ready to accept them.

1531 Only one Initialization Agent function is supported within a given PLDM subsystem. The Initialization
1532 Agent shall be implemented behind the same terminus that provides the Primary PDR Repository for the
1533 PLDM subsystem.

1534 **15.2 PLDM and power state interaction**

1535 The Initialization Agent may need to reinitialize certain sensors or termini as the result of a change of
1536 system power state. An implementation should avoid requiring the Initialization Agent to execute because
1537 of low-latency power state transitions, such as transitions between ACPI S0 and S1, or S1 and S2 states.
1538 The implementation should instead ensure that termini retain their settings across low-latency power state
1539 transitions.

1540 The Sensor Initialization PDRs include a field that tells the Initialization Agent upon which system
1541 transitions a given sensor should be initialized.

1542 **15.3 RunInitAgent command**

1543 PLDM does not specify a particular mechanism for an implementation to use to detect when to run the
1544 Initialization Agent function. For example, it does not specify how a management controller would detect a
1545 system hard reset or power-up transition. In some implementations, it will be useful to have another
1546 management controller, system firmware, or another entity decide that the Initialization Agent should run.
1547 For example, system firmware may decide that the Initialization Agent should be run after a BIOS update.

1548 To enable this, PLDM defines a RunInitAgent command that can be used to launch the Initialization Agent
 1549 “on demand.” The command includes a parameter that can select a subset of Sensor Initialization PDRs
 1550 to be used.

1551 15.4 Recommended Initialization Agent steps

1552 The following presents an outline of the steps for an Initialization Agent in a system implementation that
 1553 includes Initialization PDRs.

- 1554 1) Stop the Event Receiver function from accepting events received from any interface but the system
 1555 (host) interface.
- 1556 2) Scan the PDR Repository for Terminus Locator PDRs. Collect a list of valid termini.
- 1557 3) For each terminus in the list, perform the following actions:
 - 1558 a) Turn off Event Generation by using the SetEventReceiver command. If a terminus responds to
 1559 the SetEventReceiver command, add the terminus to a list of termini to have events re-enabled
 1560 later.
 - 1561 b) Use the GetTID command to determine whether the terminus has a TID. If so, leave that value
 1562 unchanged unless it is already assigned to another terminus. If not, use the SetTID command to
 1563 assign a TID to the terminus.
 - 1564 c) Scan the PDR Repository for Initialization PDRs (for example, numeric sensor/effector
 1565 initialization PDRs or state sensor/effector initialization PDRs) that are associated for the
 1566 terminus. For each PDR that is found, perform the following actions:
 - 1567 – Set the sensor type, sensor thresholds, and hysteresis as directed by the PDR using the
 1568 SetSensorThresholds and SetSensorHysteresis commands.
 - 1569 – Use the appropriate enabling command (for example, SetNumericSensor Enables if the
 1570 sensor is a numeric sensor) to enable scanning and event generation per the PDR.
- 1571 4) Enable the Event Receiver function to accept or poll for event messages.
- 1572 5) For each terminus with a Terminus Locator PDR, enable synchronous or asynchronous event
 1573 message generation using the SetEventReceiver command or leave it disabled (This is done at the
 1574 discretion of the Management Controller.) For each of these termini, configure an event message
 1575 transfer size via the EventMessageBufferSize command.

1576 16 Terminus and event commands

1577 This clause describes the commands that are used by PLDM termini that implement PLDM monitoring
 1578 and control as defined in this specification. The command numbers for the PLDM messages are given in
 1579 clause 30.

1580 If a PLDM terminus is implemented to provide access to any of the capabilities of this specification, the
 1581 Mandatory/Conditional (M/C) requirements shown in Table 10 apply.

1582

Table 10 – Terminus and event commands

Command	M/C	Reference
SetTID (see DSP0240)	M	See 16.1.
GetTID (see DSP0240)	M	See 16.2.
GetTerminusUID	C ^[1]	See 16.3.
SetEventReceiver	C ^{[2][3]}	See 16.4.

Command	M/C	Reference
GetEventReceiver	C ^[2]	See 16.5.
PlatformEventMessage	C ^[2]	See 16.6.
PollForPlatformEventMessage	C ^[2]	See 16.7
EventMessageSupported	C ^[4]	See 16.8
EventMessageBufferSize	C ^[4]	See 16.9

- 1583 ^[1] See 16.3.
- 1584 ^[2] Support for at least one of PlatformEventMessage or PollForPlatformEventMessage is
- 1585 mandatory for termini that generate PLDM Event Messages.
- 1586 ^[3] Sending the SetEventReceiver command is Mandatory for termini that implement the
- 1587 Initialization Agent function.
- 1588 ^[4] Mandatory for termini that generate redfishTaskExecutedEvent, redfishMessageEvent, or
- 1589 heartbeatTimerElapsedEvent class PLDM Event Messages.

1590 The following table details the classes of PLDM events supported in this specification:

1591 **Table 11 – PLDM Event Types**

PLDM Event Class	Event Class Name	Description
00h	sensorEvent	Events related to PLDM numeric and state sensors. See Table 19.
01h	effectorEvent	Events related to PLDM effectors. See Table 20.
02h	redfishTaskExecutedEvent	Events triggered by completion of long running tasks spawned by execution of RDE Operations as defined in DSP0218. See Table 21.
03h	redfishMessageEvent	Events triggered to transmit Redfish Events. See Table 22.
04h	pldmPDRRepositoryChgEvent	Events triggered by changes to the repository of PDRs. See Table 23.
05h	pldmMessagePollEvent *	This event indicates that the terminus FIFO contains a large message that will require a multipart transfer via the PollForPlatformEvent command. See Table 25.
06h	heartbeatTimerElapsedEvent *	This event indicates that a keepalive heartbeat timer has elapsed in the terminus. See Table 26.
07..EFh	reserved	reserved for future use
F0 .. FEh	oemEvent	An OEM-specific event in a format not described in this specification.
FFh	reserved	reserved for future use

1592 * These events shall only be sent asynchronously (via the PlatformEventMessage command) from the terminus. If the
 1593 terminus is configured for synchronous events (via the SetEventReceiver command), it shall not send these events.

1594 **16.1 SetTID command**

1595 The SetTID command is used to set the TID for a PLDM terminus. This command is typically used by the
 1596 PLDM Discovery Agent function. This command is defined in [DSP0240](#).

1597 **16.2 GetTID command**

1598 The GetTID command is used to retrieve the present TID setting for a PLDM terminus. This command is
 1599 defined in [DSP0240](#).

1600 **16.3 GetTerminusUID command**

1601 The GetTerminusUID command is used to obtain a unique ID for the terminus when it is necessary to
 1602 differentiate between different instances of identical devices that hold the terminus (such as two otherwise
 1603 identical add-in cards), or when it is necessary to track a particular terminus that may be “relocated,” such
 1604 as a terminus on an add-in card that is moved from one slot to another.

1605 The GetTerminusUID command shall be supported by a terminus when the terminus is on a hot-
 1606 pluggable or other add-in card where the platform management subsystem implementation is expected to
 1607 discover and automatically adopt PLDM capabilities in the terminus (such as sensors) without requiring
 1608 separate configuration steps to be taken outside of PLDM. See 14.3 and 14.2 for more information.

1609 If more than one terminus is on the same card, only the terminus that provides PDRs for the add-in card
 1610 is required to support the GetTerminusUID command. Table 12 describes the format of the command.

1611 **Table 12 – GetTerminusUID command format**

Type	Request data
–	none
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
UUID	UUIDValue

1612 **16.4 SetEventReceiver command**

1613 The SetEventReceiver command is used to set the address of the Event Receiver into a terminus that
 1614 generates event messages. It is also used to globally enable or disable whether event messages are
 1615 generated from the terminus. This version of the specification provides a polling mechanism. There shall
 1616 be a maximum of one event receiver as described in 13.2 PLDM Event Receiver. This command shall be
 1617 executed on the specific medium (binding) where the event receiver is listening. The requestor is allowed
 1618 to change the medium to transport the events by reissuing this command.

1619 The event originator (terminus) will receive the request to enable legacy asynchronous event message,
 1620 enable polling of event messages or disable all event message generation. This command permits only
 1621 one eventMessageGlobalEnable enumeration and is superseded by subsequent invocations of this
 1622 command. This specification has added additional completion codes to allow the terminus to indicate its
 1623 capabilities. While this causes the requestor to reiterate the command to determine support, the method
 1624 preserves backward compatibility to previous specifications.

1625 Table 13 describes the format of the command.

1626 **Table 13 – SetEventReceiver command format**

Type	Request data	
enum8	eventMessageGlobalEnable	
	This value is used to enable or disable event message generation from the terminus.	
	Values:	Definitions
	disable	Disable all event message generation from the terminus. The transportProtocolType and eventReceiverAddressInfo fields must be populated in the request, but shall be ignored by the receiver of this command.
	enableAsync	<p>Enable asynchronous event message generation from the terminus. This setting is combined with the enable and disable settings for individual sensors, effecters, and so on. For example, both this global enable and the individual enable for a sensor must be set to “enable” for event messages to be generated for the sensor.</p> <p>Globally enabling event generation causes all sensors and effecters within the terminus to evaluate their event state and the terminus will generate event messages if sensors’ or effecters’ present state does not match their default initialization state. Additional events (such as PDR or Redfish events) may be generated independent of the status of sensors and effecters.</p> <p>When enableAsync is chosen, the Event Receiver may also need to poll for large multipart event messages.</p>
	enablePolling	Similar to the enableAsync, the sensors and effecters will generate event messages if their present state does not match their default initialization state. A terminus is expected to return any sensor state or threshold transitions when polled by the Event Receiver. Additional events (such as PDR or Redfish events) may be generated independent of the status of sensors and effecters; these should also be returned if generated.
enableAsyncKeepAlive	enableAsync as above plus the terminus shall periodically emit the heartbeatTimerElapsedEvent as described with the heartbeatTimer field, below.	

Type	Request data (continued)
enum8	<p>transportProtocolType</p> <p>This value is provided in the request to help the responder verify that the content of the eventReceiverAddressInfo field used in this request is correct for the messaging protocol supported by the terminus. This value is defined in DSP0245. The content of the eventReceiverAddressInfo field used in this command depends on the transportProtocolType and in some cases also the medium that the terminus is using. The command shall be rejected and an INVALID_PROTOCOL_TYPE 61 completionCode returned if the transportProtocolType is incorrect.</p>
varies	<p>eventReceiverAddressInfo</p> <p>This value is a medium and protocol-specific address that the responder should use when transmitting event messages using the indicated protocol. The format, size and specification of this field depends on the transportProtocolType. The bytes in this field may contain additional information, such as protocol version, medium type, transport binding type, and so on. For example, if the transportProtocolType is MCTP (0x00), then this is a single byte field containing the Endpoint Identifier (EID) of the Event Receiver.</p> <p>The format of this field is defined in the PLDM-to-Transport binding specification identified by the transportProtocolType field.</p> <p>If the transportProtocolType value from DSP0245 is “Vendor-specific”, the overall eventReceiverAddressInfo format is vendor-specific. However, the first field of the eventReceiverAddressInfo must be a uint32 that holds a value corresponding to the IANA Enterprise Number of the vendor or organization that has specified the format.</p>
uint16	<p>heartbeatTimer</p> <p>Amount of time in seconds after each elapsing of which the terminus shall emit a heartbeat event (the heartbeatTimerElapsedEvent) to the event receiver. If the terminus cannot produce heartbeat events at the requested rate, it shall return completion code HEARTBEAT_FREQUENCY_TOO_HIGH.</p> <p>This field is mandatory if eventMessageGlobalEnable above is set to enableAsyncKeepAlive. This field shall be omitted from the request data if eventMessageGlobalEnable is set to any other value. (This preserves backward compatibility with previous versions of this specification.)</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, INVALID_PROTOCOL_TYPE=0x80, ENABLE_METHOD_NOT_SUPPORTED=0x81, HEARTBEAT_FREQUENCY_TOO_HIGH = 0x82 }</p> <p>If the requested method in eventMessageGlobalEnable is not supported, the terminus shall respond with ENABLE_METHOD_NOT_SUPPORTED. The MC may retrieve a list of supported methods via the EventMessageSupported command (clause 16.8).</p>

1627 **16.5 GetEventReceiver command**

1628 The GetEventReceiver command is used to verify the values that were set into an Event Generator using
 1629 the SetEventReceiver command. Table 14 describes the format of the command.

1630 **Table 14 – GetEventReceiver command format**

Type	Request data
–	none
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
enum8	transportProtocolType This value indicates the transportProtocolType that the terminus uses for its eventReceiverAddress and the format of the eventReceiverAddress field. This value is defined in DSP0245 .
varies	eventReceiverAddress This value is a medium and protocol-specific address that the responder should use when transmitting event messages using the indicated protocol. The format and specification of this field depends on the protocolType. The bytes in this field may contain additional information, such as protocol version, medium type, transport binding type, and so on. The format of this field is defined in the PLDM-to-Transport binding specification identified by the transportProtocolType field. If the transportProtocolType value from DSP0245 is "Vendor-specific", the overall eventReceiverAddress format is vendor-specific. However, the first field of the eventReceiverAddress must be a uint32 that holds a value corresponding to the IANA Enterprise Number of the vendor or organization that has specified the format. The value in the eventReceiverAddress field is unspecified if the eventReceiverAddress has not yet been initialized. Otherwise, the field returns the last value that was set using the SetEventReceiver command.

1631 **16.6 PlatformEventMessage command**

1632 PLDM Event Messages are sent as PLDM request messages to the Event Receiver using the
 1633 PlatformEventMessage command. Because PLDM requests have associated responses, this approach
 1634 provides a positive acknowledgement that the event message was received. Table 15 describes the
 1635 format of the command.

1636 When the terminus supplies a pldmMessagePollEvent, this indicates to the Event Receiver that the event
 1637 data is large and must be retrieved via a series of multi-part transfers using the
 1638 PollForPlatformEventMessage command. An example of this message flow may be found in clause 16.7.

1639 The formatVersion field shall be fixed at 0x01 for this format.

1640 **Table 15 – PlatformEventMessage command format**

Type	Request data	
uint8	formatVersion Version of the event format (the format and definition of the following bytes): 0x01 for the format detailed in this specification.	
uint8	TID Terminus ID for the terminus that originated the event message	
uint8	eventClass The class of event being sent. See Table 11 for a list of event types.	
var	eventData Event data based on the eventClass	
Type	Response data	
enum8	completionCode value: { PLDM_BASE_CODES, UNSUPPORTED_EVENT_FORMAT_VERSION = 0x81 }	
enum8	Status	
	Value	Definition
	noLogging	The event message has been accepted. The implementation does not provide a PLDM Event Log at the Event Receiver.
	loggingDisabled	The event message was accepted but will not be logged because logging is disabled.
	logFull	The event message was accepted but will not be logged because the log is full.
	acceptedForLogging	The event message has been accepted and queued up for logging. Note that under some conditions the message may not be logged if the log becomes full or is disabled before the queued message is processed.
	logged	The event message was accepted. The implementation has confirmed that the event has been logged prior to sending the response.
loggingRejected	The implementation has accepted the event message but has rejected logging it based on filtering of the event message content.	

1671 **16.7 PollForPlatformEventMessage command**

1642 The PollForPlatformEventMessage command enables the Event Receiver to poll for events from a PLDM
1643 terminus and acknowledge the receipt of the event message. The SetEventReceiver command enables
1644 polling of event messages if the PLDM terminus supports this command. PollForPlatformEventMessage
1645 command format is described in Table 16. This command is optional for this version of this specification.

1646 This command shall be the only method for retrieving large event messages from a terminus. This
1647 command provides a multiple part transfer mechanism to retrieve event messages, which have variable
1648 data fields. Large messages are broken into chunks of data, the size of which shall be negotiated through
1649 the EventMessageBufferSize command. An example of such a message is the
1650 pldmPDRRepositoryChgEvent.

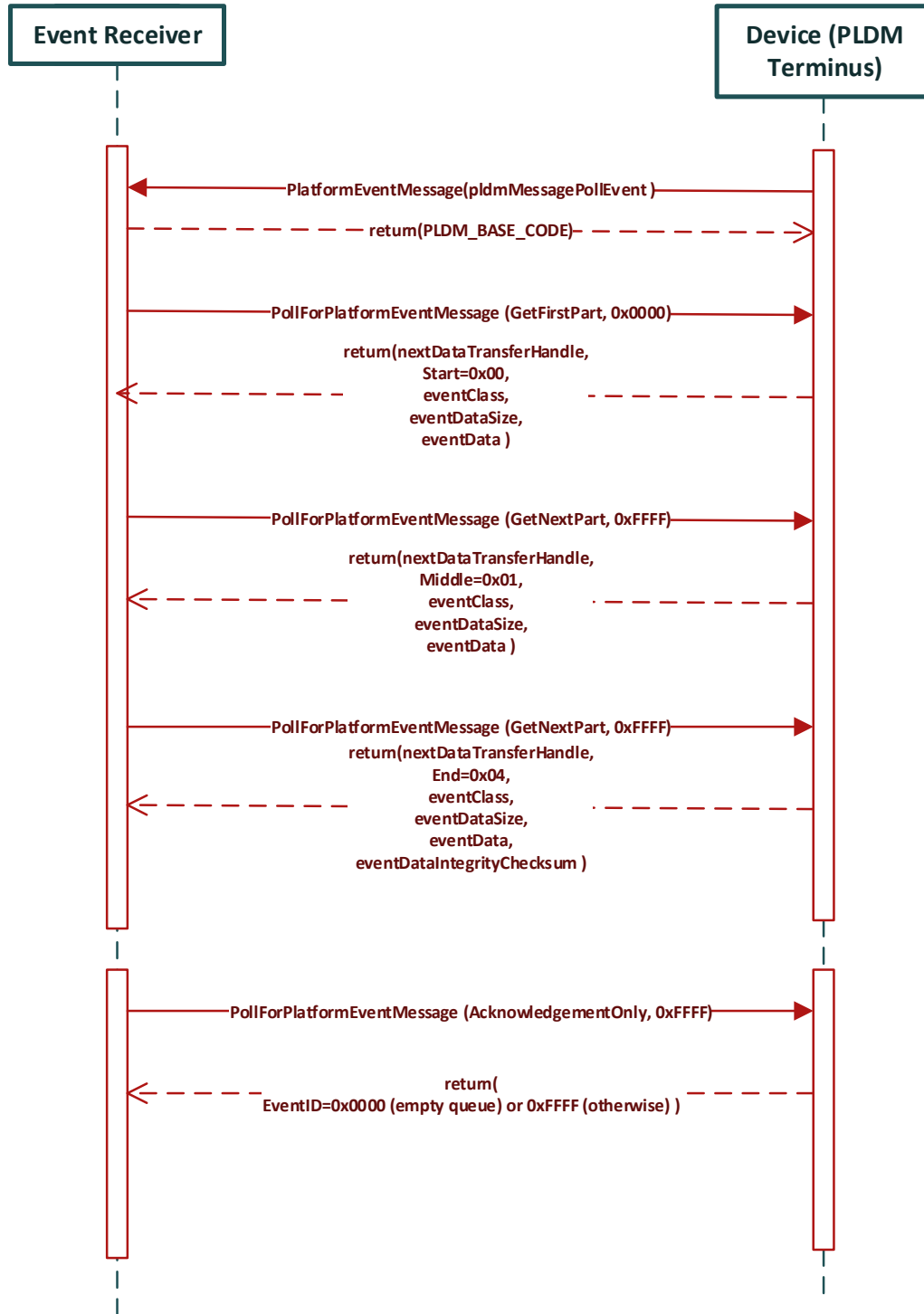
1651 Only one event is returned on each requested poll cycle and is acknowledged by the requestor on the
1652 next command invocation. The eventIDToAcknowledge shall be set to 0x0000 when retrieving the first
1653 unacknowledged event message (as determined by the terminus). This could be an event message
1654 previously returned if that message was never acknowledged. The PLDM terminus shall return an
1655 eventID greater than 0x0000 if an event is available; otherwise, eventID 0x0000 shall be returned to
1656 indicate the terminus event queue is empty. The PLDM Event Receiver shall acknowledge reception of
1657 the event by issuing the command again with the eventIDToAcknowledge set to the previously retrieved
1658 eventID (from the PLDM terminus). The PLDM terminus shall remove the acknowledged event message
1659 from its internal FIFO upon reception of the acknowledgement. The eventClass and eventData fields are
1660 not present when the eventID field is set to 0x0000 or 0xFFFF or if the completionCode is not set to
1661 SUCCESS. The recommended operation is for the PLDM Event Receiver to retrieve all messages from
1662 the terminus (e.g., poll until the PLDM terminus returns an eventID equal to 0x0000). The PLDM terminus
1663 may overwrite the oldest event message in its internal FIFO should events occur faster than the PLDM
1664 Event Receiver polls and the FIFO fills up.

1665 In the event that the Event Receiver wishes to suspend polling while more events remain to be retrieved,
1666 it may do so by issuing a final invocation of this command, with TransferOperationFlag set to
1667 AcknowledgementOnly, to acknowledge the last event it has received and processed. The Event
1668 Receiver may use this technique to stop polling for PLDM events in the case of asynchronous message
1669 transfer (via PlatformEventMessage commands originated from the terminus).

1670 If an event is sent in asynchronous mode and the terminus is switched to polling mode before the Event
1671 Receiver acknowledges the event, then the terminus shall send the oldest event on the next polling
1672 request unless the terminus overwrites the event.

1673 The formatVersion field shall be fixed at 0x01 for this specification.

1674 Figure 22 shows an example flow that demonstrates switching to polled event transfer to receive an event
1675 with large event data. When the Event Receiver gets a pldmMessagePollEvent, this is a signal that an
1676 event with a large amount of event data is next to be transferred. The Event Receiver then uses the
1677 PollForPlatformEventMessage command with TransferOperationFlag set to GetFirstPart to initiate the
1678 transfer. In response, the terminus supplies the first chunk of data along with a transfer handle for the
1679 next portion and a transferFlag of Start, which indicates that this is the first chunk and there is at least one
1680 more. The Event Receiver then retrieves the next chunk in the same fashion, using the
1681 nextDataTransferHandle supplied in the previous response. So long as the response message
1682 transferFlag field is set to Middle, the Event Receiver knows that more data is waiting to be retrieved, and
1683 repeats this process using the most recently received nextDataTransferHandle to obtain the next data
1684 chunk each time. Finally, when the transferFlag comes back as End, the Event Receiver knows the
1685 transfer is complete and can verify the eventDataIntegrityChecksum against the reassembled event data.
1686 Assuming the transfer was successful, the Event Receiver can now acknowledge receipt of the event and
1687 switch back to asynchronous transfer of events by sending a final PollForPlatformEventMessage
1688 command with TransferOperationFlag set to AcknowledgementOnly.



1689

1690 **Figure 22: Switching from asynchronous eventing to poll for an event with large data**

1691

Table 16 – PollForPlatformEventMessage command format

Type	Request data
uint8	<p>formatVersion</p> <p>Version of the event format (the format and definition of the following bytes): 0x01 for this specification.</p>
enum8	<p>TransferOperationFlag</p> <p>The operation flag that indicates whether this is the start of the transfer. Possible values: {GetNextPart=0x00, GetFirstPart=0x01, AcknowledgementOnly=0x02}</p>
uint32	<p>dataTransferHandle</p> <p>A handle that is used to identify a package data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to GetFirstPart or AcknowledgementOnly.</p>
uint16	<p>eventIDToAcknowledge</p> <p>An event previously received that should be acknowledged; The MC shall use the null value 0x0000 when requesting the first entry from the terminus' event queue. The MC shall use the special value 0xFFFF when in the middle of a multipart event transfer (TransferOperationFlag is GetNextPart)</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, UNSUPPORTED_EVENT_FORMAT_VERSION = 0x81, EVENT_ID_NOT_VALID=0x82 }</p>
uint8	<p>TID</p> <p>Terminus ID for the terminus from which event messages are being supplied</p>
uint16	<p>eventID</p> <p>The Event ID for the returned event in this response. The terminus assigns the Event ID to an event so the requester can acknowledge it on the next invocation of this command. The terminus shall supply a value of 0x0000 if the terminus internal event queue is empty. If TransferOperationFlag in the request message was set to AcknowledgementOnly and the event queue is non-empty, the terminus shall supply special value 0xFFFF for this field.</p>
uint32	<p>nextDataTransferHandle</p> <p>A handle that is used to identify the next portion of the transfer. This field shall be omitted if eventID is 0x0000 or 0xFFFF.</p>
enum8	<p>TransferFlag</p> <p>The transfer flag that indicates what part of the transfer this response represents. Possible values: {Start=0x00, Middle=0x01, End=0x04, StartAndEnd=0x05} This field shall be omitted if eventID is 0x0000 or 0xFFFF.</p>
uint8	<p>eventClass</p> <p>The type of event being returned. See Table 11 for a list of event types. This field shall be omitted if eventID is 0x0000 or 0xFFFF.</p>
uint32	<p>eventDataSize</p> <p>The size in bytes of the eventData field below. (Does not include eventDataIntegrityChecksum.) This field shall be omitted if eventID is 0x0000 or 0xFFFF.</p>

Type	Response data (continued)
var	<p>eventData</p> <p>A chunk of Event data, based on the eventClass, in a buffer sized as negotiated in the EventMessageBufferSize command.</p> <p>This field shall be omitted if eventID is 0x0000 or 0xFFFF.</p>
uint32	<p>eventDataIntegrityChecksum</p> <p>32-bit CRC for the entirety of event data (all parts concatenated together, excluding this checksum). This field shall be omitted except for final chunks of event messages containing multiple parts (TransferFlag = End).</p> <p>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 (see clause 16.9), the DataIntegrityChecksum shall be sent as the only data in another chunk (with eventDataSize set to zero).</p> <p>For this command, 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>

1692

1693 **16.8 EventMessageSupported Command**

1694 The EventMessageSupported command is optional for this specification version. It is recommended,
 1695 however, that a terminus supports this command if the terminus accepts the SetEventReceiver command.
 1696 This command returns a list of eventClass supported by the terminus. The enumeration values for the
 1697 eventClass are defined in Table 11.

1698

Table 17 – EventMessageSupported command format

Type	Request data
uint8	<p>formatVersion</p> <p>Version of the event format (the format and definition of the following bytes):</p> <p>0x01 for this specification version</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, UNSUPPORTED_EVENT_FORMAT_VERSION = 0x81 }</p>
enum8	<p>synchronyConfiguration</p> <p>This value indicates the messaging style most recently configured via the SetEventReceiver command:</p> <p>value: { NOT_CONFIGURED = 0x00, // SetEventReceiver command not received ASYNCHRONOUS_MESSAGING = 0x01, // Asynchronous messaging SYNCHRONOUS_MESSAGING = 0x02 // Poll-based messaging ASYNCHRONOUS_WITH_HEARTBEAT = 0x03 // Asynchronous messaging, heartbeat }</p>

Type	Response data (continued)
bitfield8	<p>synchronyConfigurationSupported</p> <p>This value indicates the event messaging styles supported by the terminus. For each bit, a value of 1b shall indicate that the mode is supported.</p> <p>[7:4] - Reserved for future use</p> <p>[3] - Asynchronous messaging with heartbeat</p> <p>[2] - Synchronous (poll-based) messaging</p> <p>[1] - Asynchronous messaging, no heartbeat</p> <p>[0] - Reserved; shall be 0b.</p>
uint8	<p>numberEventClassReturned</p> <p>The count N of eventClass enumerated bytes returned in this response</p>
uint8	<p>eventClass [0]</p> <p>The first eventClass message the device can generate. The eventClass values are defined in Table 11.</p>
uint8	<p>eventClass [1]</p> <p>The second eventClass message the device can generate. The eventClass values are defined in Table 11.</p>
uint8	...
uint8	<p>eventClass [N-1]</p> <p>The last eventClass message the device can generate. The eventClass values are defined in Table 11.</p>

1699

1700 **16.9 EventMessageBufferSize Command**

1701 The EventMessageBufferSize command is optional for this specification version. It is recommended,
 1702 however, a terminus supports this command if the terminus accepts the SetEventReceiver command.
 1703 This command communicates the maximum size of the event receiver buffer that can hold a single event
 1704 message. The response is the maximum size of the terminus buffer that can transmit a single event
 1705 message. The smaller of the two values shall be the negotiated event message size. Any event message
 1706 that exceeds the negotiated event message buffer size shall be retrieved by the event receiver using the
 1707 PollForPlatformEventMessage command. The terminus shall send the pldmMessagePollEvent to the
 1708 PLDM event receiver when an event message exceeds the negotiated buffer size.

1709 In the event that this command is not invoked, a default message buffer size of 256 bytes shall be in
 1710 effect.

1711 **Table 18 – EventMessageBufferSize command format**

Type	Request data
uint16	<p>eventReceiverMaxBufferSize</p> <p>This is the maximum buffer to hold an event message transferred from the terminus to the event receiver.</p>

Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
uint16	terminusMaxBufferSize This is the maximum size of an event message sent from the terminus to the event receiver. The smaller of eventReceiverMaxBufferSize and terminusMaxBufferSize shall be the negotiated size for all event messages regardless of asynchronous or polled.

1712 **16.10 eventData format for sensorEvent**

1713 Table 19 defines the format of the eventData field in PLDM Event Messages for the sensorEvent class.
 1714 This field includes event data for PLDM state sensor and numeric sensor events, and for events related to
 1715 changes of the sensor's operational state.

1716 **Table 19 – sensorEvent class eventData format**

Type	Request data
uint16	sensorID The sensorID is the value that is used in PDRs and PLDM sensor access commands to identify and access a particular sensor within a terminus.
enum8	sensorEventClass value: { sensorOpState, // Events from a PLDM state or numeric sensor that are related to // changes of the sensor's operational state stateSensorState, // Events from a PLDM state sensor that are related to a change // in the present state from the set of states that the sensor is // monitoring numericSensorState // Events from a PLDM numeric sensor that are related to a change // in the present state from the set of states that the sensor is // monitoring. Also returns the reading value that triggered the event. } <i>For sensorEventClass = stateSensorState</i>
uint8	sensorOffset Identifies which state sensor within a composite state sensor the event is being returned for. 0x00 = first state sensor, 0x01 = second state sensor, and so on value: 0x00 to 0x07
enum8	eventState The event state value from the state change that triggered the event message. See Table 41 for the definition of eventState.

Type	Request data (continued)
enum8	<p>previousEventState</p> <p>The event state value for the state from which the present event state was entered.</p> <p>See Table 41 for the definition of eventState.</p> <p>special value: This value shall be set to the same value as eventState if the previous event state is unknown, which may be the case for events that are generated on the first status assessment that occurs after a sensor has been initialized.</p>
<i>For sensorEventClass = numericSensorState</i>	
enum8	<p>eventState</p> <p>The eventState value from the state change that triggered the event message.</p> <p>See Table 30 for the enumeration values of eventState.</p>
enum8	<p>previousEventState</p> <p>The eventState value for the state from which the present state was entered.</p> <p>See Table 30 for the enumeration values of eventState.</p> <p>special value: This value shall be set to the same value as eventState if the previous event state is unknown (which may be the case for events that are generated on the first status assessment that occurs after a sensor has been initialized).</p>
enum8	<p>sensorDataSize</p> <p>The bit width and format of reading and threshold values that the sensor returns</p> <p>value: { uint8, sint8, uint16, sint16, uint32, sint32 }</p>
uint8 sint8 uint 16 sint16 sint32 uint32	<p>presentReading</p> <p>The present value indicated by the sensor. The sensorDataSize field returns an enumeration that indicates the number of bits used to return the value.</p>
<i>For sensorEventClass = sensorOpState</i>	
enum8	<p>presentOpState</p> <p>The sensorOperationalState value from the state change that triggered the event message.</p> <p>See Table 30 for the enumeration values of sensorOperationalState.</p>
enum8	<p>previousOpState</p> <p>The sensorOperationalState value for the state from which the present state was entered.</p> <p>See Table 30 for the enumeration values of sensorOperationalState.</p> <p>special value: This value shall be set to the same value as presentOpState if the previousOpState is unknown, which may be the case for events that are generated on the first status assessment that occurs after a sensor has been initialized.</p>

1717 **16.11 eventData format for effectorEvent**

1718 Table 20 defines the format of the eventData field in PLDM Event Messages for the effectorEvent class.

1719 This field supports events for changes of the effector's operational state.

1720

Table 20 – effectorEvent class eventData format

Type	Request data
uint16	<p>effectorID</p> <p>The effectorID is the value that is used in PDRs and PLDM effector access commands to identify and access a particular effector within a terminus.</p>
enum8	<p>effectorEventClass</p> <p>value: {</p> <p style="padding-left: 40px;">effectorOpState // Events from a PLDM state or numeric effector that are related to changes of the effector's operational state</p> <p>}</p>
<i>For effectorEventClass = effectorOpState</i>	
enum8	<p>presentOpState</p> <p>The effectorOperationalState value from the state change that triggered the event message.</p>
enum8	<p>previousOpState</p> <p>The effectorOperationalState value for the state from which the present state was entered.</p> <p>special value: This value shall be set to the same value as presentOpState if the previousOpState is unknown, which may be the case for events that are generated on the first status assessment that occurs after an effector has been initialized.</p>

1721 **16.12 eventData format for redfishTaskExecutedEvent**

1722 Table 21 defines the format of the eventData field in PLDM Event Messages for the redfishTaskExecuted
 1723 class. This field supports PLDM events for completion of a long-running Redfish Task as defined in
 1724 [DSP0218](#).

1725

1726 **16.12.1 Table 21 – redfishTaskExecutedEvent class eventData format**

Type	Request data
uint32	<p>resourceID</p> <p>The ResourceID is the value that is used in PDRs and PLDM for Redfish Device Enablement commands to identify and access a particular collection of schema-based Redfish data</p>
uint16	<p>operationID</p> <p>Operation associated with the Task that has completed execution</p>

1727

1728 **16.13 eventData format for redfishMessageEvent**

1729 Table 22 defines the format of the eventData field in PLDM Event Messages for the redfishMessageEvent
 1730 class. A PLDM event may contain one or more Redfish Events. See [DSP0218](#) for information on how
 1731 PLDM for Redfish Device Enablement uses RDE events and [DSP0266](#) for information on the events
 1732 themselves.

1733 Redfish Events contain timestamps. For RDE Devices that do not contain realtime clocks, the timestamp
 1734 shall be set to a sentinel value of zero. When decoding Redfish Events with the timestamp set to the zero
 1735 sentinel, the MC may substitute a current timestamp.

1736 **Table 22 – redfishMessageEvent class eventData format**

Type	Request data
uint8	eventCount The number of Redfish Events N encoded in the eventData field below.
uint16	eventDataLength Length in bytes of the eventData field below, which comprises the encoding of one or more Redfish Events contained within this PLDM event. This value shall not exceed the negotiated event message size.
uint32	resourceID [0] An opaque handle referencing the particular collection of schema-based Redfish data associated with the first Redfish Event encoded in the eventData field below.
enum8	eventSeverity [0] The severity of the first Redfish Event in the Redfish EventRecords array encoded in eventData below. Value = {OK = 0, Warning = 1, Critical = 2}
...	...
uint32	resourceID [N - 1] An opaque handle referencing the particular collection of schema-based Redfish data associated with the last Redfish Event encoded in the eventData field below.
enum8	eventSeverity [N - 1] The severity of the last Redfish Event in the Redfish EventRecords array encoded in eventData below. Value = {OK = 0, Warning = 1, Critical = 2}
bejEncoding	eventData BEJ encoded Event payload data. The bejEncoding PLDM type is defined in DSP0218 .

1737 **16.14 eventData format for pldmPDRRepositoryChgEvent**

1738 This Event is to signal the PLDM Event Receiver that there is a change in the terminus PDR repository.
 1739 The device will return the PDR Types or the PDR Record Handles for the PDRs to be retrieved from the
 1740 terminus. This allows a simple method for a terminus to indicate which portion of its “virtual” PDR
 1741 Repository needs to be refreshed. The PLDM terminus client (or event receiver) will need to comprehend
 1742 additions, deletions and modifications of the PDRs as it updates the system primary PDR repository. The
 1743 terminus may indicate the entire repository is to be retrieved by setting the eventDataFormat to a special
 1744 value of “refreshEntireRepository”. The terminus shall not mix “PDR Types” and “PDR Record Handles” in
 1745 a single event message.

1746 The terminus may have multiple operations in each event message but the operations shall be sent in the
 1747 following sequence:

- 1748 1. PDR records to be removed (deleted) from the event receiver’s repository shall be first, grouped
1749 either in a single event message or as individual event messages.
- 1750 2. PDR records to be added to the event receiver’s repository shall be after the deleted records,
1751 grouped either in a single event message or as individual event messages.
- 1752 3. The existing PDR records to be modified in the event receiver’s repository shall be last, grouped
1753 either in a single event message or as individual event messages.

1754 For example, if a hard drive is added to a storage enclosure under control of an intelligent storage
1755 adapter, the terminus could indicate the addition of PDRs representing the newly added hard drive in one
1756 event message followed by another event message indicating the affected Entity Association PDRs. The
1757 event receiver, which may also be the primary repository manager, only needs to retrieve the affected
1758 PDRs rather than the entire repository.

1759 Another example is if an entire storage enclosure is removed, the number of affected PDRs returned in
1760 this event message may exceed the MCTP baseline transmission unit size. In this example, setting the
1761 eventDataFormat to a special value of “refreshEntireRepository” is the best choice.

1762 The goal of this event is to avoid retrieving the entire device PDR repository for a small device PDR
1763 repository differences.

1764

1765 **Table 23 – pldmPDRRepositoryChgEvent class eventData format**

Type	Request data
enum8	eventDataFormat { refreshEntireRepository, formatIsPDRTypes, formatIsPDRHandles } This field indicates if the changedRecords are of PDR Types or PDR Record Handles. The device may signal to the event receiver to re-enumerate the entire device PDR repository by supplying the value refreshEntireRepository. To signal that only certain types of PDRs should be refreshed, the device shall supply the value formatIsPDRTypes and provide one change record below for each type of PDR to be refreshed.
uint8	numberOfChangeRecords The number of changeRecords N_R following this field. If the eventDataFormat is refreshEntireRepository, this value shall be zero.
var	changeRecord [0] See Table 24 – pldmPDRRepositoryChgEvent changeRecord format for details. This field is not present if the numberOfChangeRecords is zero.
var	changeRecord [1]
...	...
var	changeRecord [$N_R - 1$]

1766

1767

Table 24 – pldmPDRRepositoryChgEvent changeRecord format

Type	Request data
enum8	<p>eventDataOperation { refreshAllRecords, recordsDeleted, recordsAdded, recordsModified }</p> <p>For each pldmPDRRepositoryChgEvent record, there can only be a single operation. This simplifies the parsing for both the terminus and the event receiver. The order the event records are provided shall be “RefreshAll”, “Deleted”, “Added”, “Modified”.</p> <p>The value refreshAllRecords shall only be supplied when eventDataFormat was set to formatIsPDRTypes. In this case, the entries below represent a series of PDR types to be refreshed.</p>
uint8	<p>numberOfChangeEntries</p> <p>The number of change entries N_E following this field.</p>
uint32	<p>changeEntry [0]</p> <p>This value will be either a “PDR Type” enumeration or a “PDR Record Handle” as enumerated by the “eventDataFormat” field in the pldmPDRRepositoryChgEvent event message.</p> <p>There may be multiple PDR Types (such as Numeric Sensor, State Sensor and Entity Association Sensor) to be retrieved due to a “hot-plug” event for the terminus. All the changed PDR Types may be returned in a single event message. The client (or event receiver) can use the FindPDR command to gather the PDR record.</p> <p>Alternatively, the terminus may provide a list of PDR Record Handles, which the MC can use as input to the GetPDR command.</p>
uint32	changeEntry [1]
...	...
uint32	changeEntry [$N_E - 1$]

1768

1769 **16.15 eventData format for pldmMessagePollEvent**

1770 Table 25 defines the format of the eventData field in PLDM Message Poll Event. This event typically
 1771 signals the event receiver that a polling command is needed to retrieve a large event message from the
 1772 terminus.

1773

Table 25 – pldmMessagePollEvent class eventData format

Type	Request data
uint8	<p>formatVersion</p> <p>Version of the event format (the format and definition of the following bytes):</p> <p>0x01 for this specification.</p>
uint16	<p>eventID</p> <p>Identifier for the event that requires multipart transfer.</p>
uint32	<p>dataTransferHandle</p> <p>A handle that is used to identify the event data to be received via the PollForPlatformEventMessage command.</p>

1774 16.16 eventData format for heartbeatTimerElapsedEvent

1775 Table 26 defines the format of the eventData field in Heartbeat Timer Elapsed Event. The terminus
1776 periodically emits this event in order to assert that the connection between itself and the MC remains
1777 active. This event shall only be emitted when the eventMessageGlobalEnable field in the
1778 SetEventReceiver command (clause 16.4) request message is set to enableAsyncKeepAlive.

1779 **Table 26 – heartbeatTimerElapsedEvent class eventData format**

Type	Request data
uint8	<p>formatVersion</p> <p>Version of the event format (the format and definition of the following bytes):</p> <p>0x01 for this specification.</p>
uint8	<p>sequenceNumber</p> <p>A sequence number for the heartbeat timer, incremented by one each time the timer elapses. This enables the MC to detect whether it has missed a heartbeat.</p>

1780 17 PLDM Numeric Sensors

1781 This clause provides information that describes the characteristics and operation of PLDM Numeric
1782 Sensors.

1783 17.1 Sensor readings, data sizes

1784 PLDM Numeric Sensors can return a present reading value. The value is returned as a binary integer.
1785 The size of this integer and whether it is signed can vary on a per-sensor basis. The PLDM
1786 GetSensorReading command includes a parameter in its response that indicates the format used for
1787 returning the reading. The same format is used for any thresholds and hysteresis values that are used for
1788 request or response parameters. Additionally, the data size is supported in PDR information for the
1789 sensor.

1790 17.2 Units and reading conversion

1791 The sensor commands do not intrinsically identify what type of unit, such as volts, amps, or RPM, is used
1792 for the sensor's present reading value. Additionally, the value may require scaling to convert the value to
1793 normalized units, such as millivolts (mV), nanoseconds, and so on.

1794 For example, microcontrollers commonly incorporate an 8-bit analog-to-digital (A/D) converter. If the
1795 converter is monitoring a signal where the 0x00 value of the conversion corresponds to 0 volts and a
1796 0xFF reading corresponds to 4.00 volts, each count of the converter corresponds to a value of $4.0/255 \approx$
1797 15.686274 mV per count. Converting a particular reading from counts into volts requires multiplying the
1798 reading by a conversion factor. A reasonable guideline is that the conversion factor should be accurate to
1799 at least 4 times the resolution of the converter. In this case, the resolution of the converter is 1 part in 255,
1800 which would require the accuracy of the conversion factor to be to better than 1 part in 1020, which
1801 rounds up to four significant digits, or 15.69 mV per count.

1802 To avoid the need for a floating point format for sensor readings and the need for multibyte multiplications
1803 and divisions in simple devices, PLDM readings are returned as “raw” integers that are converted to
1804 normalized units by the consumer of the reading data by using a specified conversion formula and
1805 sensor-specific conversion factors. The consumer of the PLDM sensor reading data will be a device
1806 serving a role such as a MAP that has more resources for doing mathematical operations. This approach
1807 avoids burdening simple devices with the conversion task.

1808 The conversion formula is specified in 27.7. The conversion factors must be provided by the vendor or
1809 designer of the particular sensor implementation. The PDR for a numeric sensor supports returning
1810 conversion factors and the type of units (volts, amps, and so on) used for a particular numeric sensor.

1811 **17.3 Reading-only or threshold-based numeric sensors**

1812 A particular instance of a PLDM Numeric Sensor can return just a numeric reading or a numeric reading
1813 *and* a threshold-based status. These sensors are referred to as "reading-only" or "threshold-based"
1814 numeric sensors.

1815 **17.4 Readable and settable thresholds**

1816 A given instance of a PLDM Numeric Sensor may have thresholds that are readable through the
1817 GetSensorThresholds command or that are settable through the SetSensorThresholds command. The
1818 PDR information can indicate whether a particular numeric sensor uses thresholds and, if so, which
1819 thresholds are supported and whether they are settable. To avoid the need for a floating point format for
1820 threshold settings and the need for multibyte multiplications and divisions in simple devices, the
1821 GetSensorThresholds and SetSensorThresholds commands must use "raw" integers to be used in the
1822 conversion formula specified in the specific numeric sensor PDR.

1823 **17.5 Update/polling intervals and states updates**

1824 A sensor may periodically collect internal readings and status (that is, it may poll for updates) and
1825 respond to a GetSensorReading request with the last collected values, or it may collect the values "on
1826 demand" upon receiving the request.

1827 An updateInterval value in the PDR for the sensor provides a way for the requester to determine the
1828 maximum time from when a sensor was re-armed or accessed to when the subsequent eventState or
1829 reading update should have occurred.

1830 For a sensor that polls for updates, the updateInterval corresponds to the nominal polling interval, $\pm 50\%$.
1831 (The $\pm 50\%$ variation is to accommodate manufacturing variations between devices implementing sensors
1832 and variations in firmware-based polling intervals.) There is no requirement for a sensor's polling interval
1833 to be synchronized (restarted) when a re-arm occurs. A sensor is also allowed to take as long as two
1834 polling intervals before updating its state following a re-arm (one interval to recognize the re-arm, and one
1835 interval to collect and apply the updated state).

1836 For a sensor that updates "on demand," the updateInterval indicates the maximum time, $\pm 50\%$, from
1837 receiving a GetSensorReading command to when a reading and status update should occur. If the sensor
1838 can update itself within the PLDM Request-to-response time (refer to [DSP0240](#)), either an updateInterval
1839 value of 0 or the actual update interval may be used in the PDR.

1840 If the updateInterval for a given sensor is longer than the PLDM Request-to-response time, the
1841 updateInterval must be specified and the sensorOperationalStatus must be returned as "initializing" while
1842 the sensor is performing its initial state assessment after being enabled or re-armed.

1843 Because a sensor is allowed to take up to two polling intervals to update after a re-arm, and because the
1844 variation is allowed to be $\pm 50\%$, it may take as long as three nominal polling intervals (two nominal
1845 intervals times 1.5) plus a PLDM Request-to-response time before the effect of a re-arm is realized.

1846 **17.6 Thresholds, Present State, and Event State**

1847 PLDM Numeric Sensors that are threshold-based have associated thresholds against which the reading
1848 is compared.

1849 **17.6.1 Threshold severity levels**

1850 Each threshold is associated with a severity that is related to how far the threshold is from the normal
 1851 range of the sensor. Unless otherwise specified, the severity level is generally based on the view that a
 1852 sensor is monitoring parameters that are associated with a physical entity. Table 27 describes the
 1853 threshold severity levels.

1854 **Table 27 – Threshold severity levels**

Severity level	Description
warning	The reading is outside of normal expected operating range but the monitored entity is expected to continue to operate normally. The warning may be an indication of a condition that is expected to become critical or fatal with time unless steps are taken to counter the condition that is causing the warning. As such, warning thresholds are usually implemented when some automated or remote action can be taken as a result of seeing the warning. For example, an application might use a warning related to an over-temperature condition to take actions to increase the system cooling or decrease its load. A warning related to increasing levels of correctable errors in a memory device might trigger an action to schedule a service call to replace the memory device before it fails.
critical	The reading is outside of supported operating range. Monitored entities might operate abnormally, have transient failures, or propagate errors to other entities under this condition. Prolonged operation under this condition might result in degraded lifetime for the monitored entity. The monitored entity will usually return to normal operation if the condition returns to a warning or normal level. A sensor reaching the critical threshold should not cause a permanent failure of the entity.
fatal	The reading is outside of rated operating range. Monitored entities might experience permanent failures or cause permanent failures to other entities under this condition. Remedial actions might require replacement of the monitored entity or other components. The reaction to the entity crossing the fatal threshold is outside the scope of this specification which may include becoming nonresponsive.

1855 **17.6.2 Upper and lower thresholds**

1856 A given threshold for a PLDM Numeric Sensor can be either an upper or a lower threshold. Upper
 1857 thresholds are for tracking events that become more severe as the reading becomes more positive
 1858 numerically. Lower thresholds are for events that become more severe as the reading becomes more
 1859 negative numerically.

1860 PLDM has three upper thresholds: upper warning, upper critical, and upper fatal. Similarly, PLDM has
 1861 three lower thresholds: lower warning, lower critical, and lower fatal. By convention, these thresholds
 1862 occur in the following order: lower fatal, lower critical, lower warning, upper warning, upper critical, and
 1863 upper fatal. Lower fatal corresponds to the most negative threshold value, and upper fatal corresponds to
 1864 the most positive threshold value. This order is illustrated in Figure 23.

1865 A sensor is not required to implement all thresholds. For example, a sensor that monitors for an over-
 1866 voltage condition may implement only an upper critical threshold. A sensor that is monitoring a low-RPM
 1867 condition may implement only lower warning and lower critical thresholds. A temperature sensor may
 1868 implement both upper and lower thresholds so that it can track both over-temperature and under-
 1869 temperature conditions.

1870 **17.6.3 Present State**

1871 A PLDM Numeric Sensor that uses thresholds returns a presentState value that is based on a simple
 1872 numeric comparison of the present reading against the sensor to the thresholds and returns the threshold
 1873 range with which the reading is associated. The presentState value is updated solely based on a numeric
 1874 comparison of the present reading to the thresholds. For upper thresholds, the presentState value is

1875 based on whether the present reading is greater than or equal to the threshold value. For lower
1876 thresholds, the presentState value is based on whether the present reading is less than or equal to the
1877 threshold value. For example, if the presentState value is greater than or equal to the value for upper
1878 critical threshold but is less than the value for upper fatal threshold, the presentState value will be
1879 UpperCritical.

1880 **17.6.4 Event State**

1881 The eventState field of a PLDM Numeric Sensor is updated based on transitions between the different
1882 monitored states of the sensor. Unlike presentState, the eventState value includes the effect of the
1883 hysteresis setting. If the hysteresis value for the sensor is equal to one count of the reading, the
1884 eventState and presentState values will be the same. Otherwise, the eventState setting may vary from
1885 the presentState due to the effect of hysteresis. See 17.9 for more information about hysteresis and its
1886 relationship to eventState.

1887 The eventState behavior is also affected by whether the sensor implementation is manual- or auto-rearm
1888 (see 17.7).

1889 **17.7 Manual re-arm and auto re-arm sensors**

1890 The event state tracking for a sensor can be either auto re-arm or manual re-arm. An auto re-arm sensor
1891 updates its eventState automatically whenever the sensor detects that a state transition has occurred.

1892 A manual re-arm sensor retains the most severe event state transition that it has detected since the time
1893 the sensor was initialized or since the last time the eventState value was explicitly cleared (using the re-
1894 arm operation in the GetSensorReading command). If a new state is assessed that has the same
1895 criticality as the previous state, the most recently assessed value shall be returned. For example, if the
1896 previous value was upperCritical and the presentState value is lowerCritical, then upperCritical shall be
1897 returned.

1898 Thus, auto re-arm sensors automatically update their status on *any* detected state transition, while
1899 manual re-arm sensors automatically update their eventState value only on detecting a worsening
1900 (increasing severity) transition (or upon a transition to a different state of equivalent severity as the
1901 previous state).

1902 Re-arming of numeric sensors is done through the GetSensorReading command. Re-arming causes the
1903 sensor to internally enter its "initializing" operating state until it next updates its presentState and
1904 eventState. (This update may happen so quickly that the temporary entry into the initializing state is never
1905 reflected in the sensorOperationalState parameter of the GetSensorReading command.)

1906 **17.8 Event message generation**

1907 A PLDM Numeric Sensor that supports and is enabled to generate event messages shall generate them
1908 whenever an Event State (eventState) change is detected. To detect changes in the Event State, the
1909 sensor implementation must do periodic polling or incorporate some other asynchronous mechanism,
1910 such as the occurrence of an interrupt, which causes the sensor to obtain a new reading, the eventState
1911 to update and an event message to be generated.

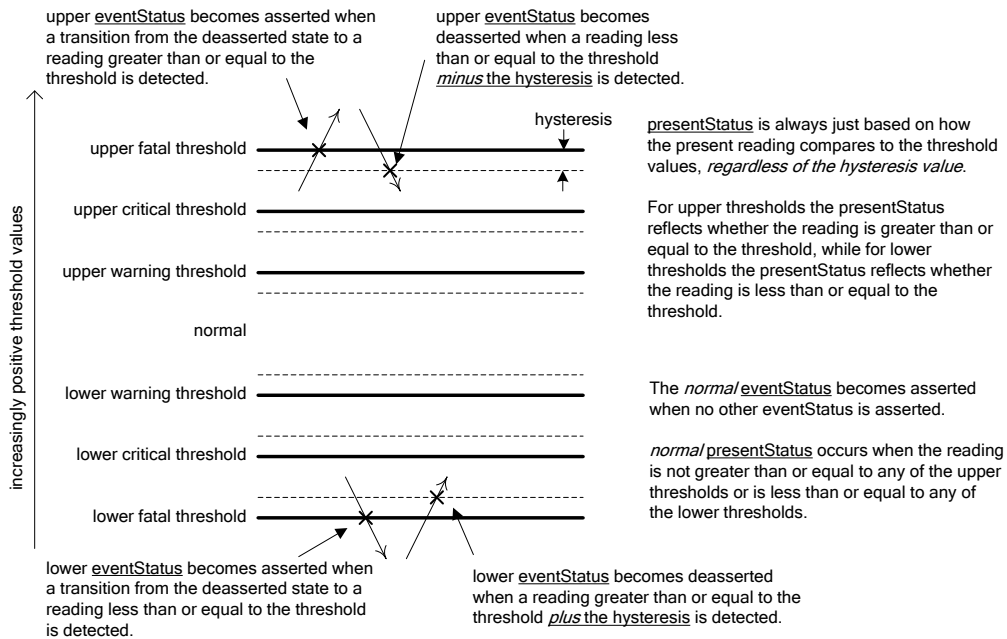
1912 **17.9 Threshold values and hysteresis**

1913 Threshold settings for PLDM Numeric Sensors are required to be ordered from numerically most negative
1914 to most positive in the following order: lower fatal, lower critical, lower warning, upper warning, upper
1915 critical, upper fatal. The hysteresis value is always subtracted from the "upper" thresholds and added to
1916 the "lower" thresholds.

1917 Thus, hysteresis is always applied on the transition from a more severe state to a less severe state. For
1918 example, assume that a sensor has a hysteresis value of 2, has an upper critical threshold set to 80, and

1919 is presently in the "upper warning" state. The sensor will transition to the "upper critical" state when it
 1920 detects that the reading value reaches a value that is greater than or equal to the threshold setting of 80.
 1921 The sensor is now in the "upper critical" state. To return to the "upper warning" state, the reading has to
 1922 drop to 78 (80 minus the hysteresis value of 2).

1923 Figure 23 helps further describe and illustrate the relationships between thresholds, hysteresis,
 1924 eventState, and presentState for numeric sensors.



1925

1926

Figure 23 – Numeric sensor threshold and hysteresis relationships

1927 **18 PLDM Numeric Sensor commands**

1928 This clause describes the commands for accessing PLDM Numeric Sensors per this specification. The
 1929 command numbers for the PLDM messages are given in clause 30.

1930 If PLDM numeric sensors are implemented, the Mandatory/Optional/Conditional (M/O/C) requirements
 1931 shown in Table 28 apply.

1932 **Table 28 – Numeric Sensor commands**

Command	M/O/C	Reference
SetNumericSensorEnable	M	See 18.1.
GetSensorReading	M	See 18.2.
GetSensorThresholds	O, C ^[1]	See 18.3.
SetSensorThresholds	O	See 18.4.
RestoreSensorThresholds	O	See 18.5.
GetSensorHysteresis	O, C ^[2]	See 18.6.
SetSensorHysteresis	O	See 18.7.
InitNumericSensor	C ^[3]	See 18.8.

1933 ^[1] The GetSensorThresholds command is required if the SetSensorThresholds command is implemented. Otherwise,
 1934 the command is optional.

1935 ^[2] The GetSensorHysteresis command is required if the SetSensorHysteresis command is implemented. Otherwise,
 1936 the command is optional.

1937 ^[3] The InitNumericSensor command is required if the sensor requires initialization following any one of the conditions
 1938 identified in the initConditions field of the PLDM Numeric Sensor Initialization PDR.

1939 **18.1 SetNumericSensorEnable command**

1940 The SetNumericSensorEnable command is used to set the operating state of the sensor itself and
 1941 whether the sensor generates event messages. Changing this state affects only the operation of the
 1942 sensor; it has no effect on the operational state of the entity or parameter that is being monitored. Event
 1943 message generation is optional for a sensor. Table 29 describes the format of the command.

1944 **Table 29 – SetNumericSensorEnable command format**

Type	Request data
uint16	sensorID A handle that is used to identify and access the sensor special values: 0x0000, 0xFFFF = reserved
enum8	sensorOperationalState The desired state of the sensor This enumeration is a subset of the operational state values that are returned by the GetSensorReading command. Refer to the GetSensorReading command for the definition of the values in this enumeration. value: { enabled, disabled, unavailable }

Type	Request data (continued)
enum8	<p>sensorEventMessageEnable</p> <p>This value is used to enable or disable event message generation from the sensor.</p> <p>value: { noChange, disableEvents, enableEvents, enableOpEventsOnly, enableStateEventsOnly}</p> <p>noChange means do not alter the present setting. Use noChange when the sensor does not support event message generation.</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, INVALID_SENSOR_ID = 0x80, INVALID_SENSOR_OPERATIONAL_STATE = 0x81, EVENT_GENERATION_NOT_SUPPORTED = 0x82 //an attempt was made to enable or disable event generation for a sensor that does not support event message generation. }</p>

1945 **18.2 GetSensorReading command**

1946 The GetSensorReading command is used to get the present reading and threshold event state values
 1947 from a numeric sensor, as well as the operating state of the sensor itself. Table 30 describes the format of
 1948 the command.

1949 NOTE The Numeric Sensor PDR sensorID type, in clause 28.4 Numeric Sensor PDR has been changed in version
 1950 1.1.1 of this specification from uint8 to uint16 to be consistent with GetSensorReading command.

1951 **Table 30 – GetSensorReading command format**

Type	Request data
uint16	<p>sensorID</p> <p>A handle that is used to identify and access the sensor</p> <p>special values: 0x0000, 0xFFFF reserved</p>
bool8	<p>rearmEventState</p> <p>true = manually re-arm EventState after responding to this request</p> <p>Re-arming causes the sensor to enter the “initializing” state until it updates its presentState and eventState.</p> <p>Sensor implementations shall either update that status immediately upon responding to this command or wait for the conclusion of their polling interval before updating the eventState.</p> <p>If event messages are enabled, the status update shall also cause the sensor to issue a corresponding assertion event message based on the eventState that it assesses. This includes generating an event message for the “normal” state.</p> <p>false = no manual re-arm</p>

1952

Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, INVALID_SENSOR_ID = 0x80, REARM_UNAVAILABLE_IN_PRESENT_STATE = 0x81 }</p>
enum8	<p>sensorDataSize</p> <p>The bit width and format of reading and threshold values that the sensor returns</p> <p>value: { uint8, sint8, uint16, sint16, uint32, sint32 }</p>
enum8	<p>sensorOperationalState</p> <p>The state of the sensor itself</p> <p>value: { enabled, disabled, unavailable, statusUnknown, failed, initializing, shuttingDown, inTest }</p> <p>enabled Enabled and operating. The sensor is able to return valid presentState, previousState, presentReading, and eventState values. This state can be set through the SetNumericSensorEnable command.</p> <p>The unavailable operational state indicates a condition in which the sensor is unable to assess one of the other state values. This typically transient condition may occur when a sensor is being initialized or has been re-armed. For the following states, the presentState, eventState, and eventDeassertionStatus values shall be set to "Unknown". Other actions related to monitoring by the sensor may also cease in this state. For example, a sensor device that polls to collect monitored values may stop polling. Unless otherwise specified, the following states are not settable through PLDM commands.</p> <p>disabled The sensor is disabled from returning presentReading and event state values. This state is settable through the SetNumericSensorEnable command.</p> <p>unavailable The sensor should be ignored due to the configuration of the platform or monitored entity. For example, the sensor is for monitoring a processor temperature, but the processor is not installed. This state is settable through the SetNumericSensorEnable command.</p> <p>statusUnknown The sensor cannot presently return valid state or reading information for the monitored entity.</p> <p>failed The sensor has failed. The sensor implementation has determined that it can not return correct values for one or more of its presentState or eventState values.</p> <p>initializing The sensor is in the process of transitioning to the operating state because the sensor is initializing (starting) or reinitializing. The presentState and eventState values shall be ignored while the sensor is in this state.</p> <p>shuttingDown The sensor is transitioning to the disabled, failed, or unavailable states.</p> <p>inTest The sensor is presently undergoing testing.</p> <p>NOTE The operation of sensor testing and the mechanisms for sensor testing are outside the scope of this specification.</p>
enum8	<p>sensorEventMessageEnable</p> <p>value: { noEventGeneration, eventsDisabled, eventsEnabled, opEventsOnlyEnabled, stateEventsOnlyEnabled }</p>

Type	Response data (continued)
enum8	<p>presentState</p> <p>The most recently assessed state value monitored by the sensor. Refer to 17.5 for additional information on how presentState is assessed.</p> <p>If the sensorOperationalState is set to enabled the sensor must return a value other than "Unknown" for the presentState.</p> <p>If the sensorOperationalState is not set to enabled the sensor shall return "Unknown" for the presentState. Parties that are using this command should also ignore the presentState value except when sensorOperationalState is set to enabled. Refer to 17.6 for important information about how presentState and eventState are generated.</p> <p>value: { Unknown, Normal, Warning, Critical, Fatal, LowerWarning, LowerCritical, LowerFatal, UpperWarning, UpperCritical, UpperFatal }</p>
enum8	<p>previousState</p> <p>The state that the presentState was entered from. This must be different from the present state (with the exception that there may be conditions where both the presentState and previousState are returned as "Unknown").</p> <p>The previousState is updated whenever the presentState is assessed as different from the previously assessed value for presentState. Refer to 17.5 for additional information on how presentState is assessed.</p> <p>If the sensorOperationalState is set to enabled the sensor may temporarily return "Unknown" for the previousState if the sensor has not yet assessed a previousState value (as may happen immediately after the sensor has become enabled). Otherwise, the sensor must return a value other than "Unknown".</p> <p>If the sensorOperationalState is not set to enabled the sensor shall return "Unknown" for the previousState. Parties that are using this command should also ignore the previousState value except when sensorOperationalState is set to enabled. Refer to 17.6 for important information about how presentState and eventState are generated.</p> <p>value: { Unknown, Normal, Warning, Critical, Fatal, LowerWarning, LowerCritical, LowerFatal, UpperWarning, UpperCritical, UpperFatal }</p>
enum8	<p>eventState</p> <p>Indicates which threshold crossing assertion events have been detected. The sensor is required to return one of the specified values in the enumeration. However, the value is required to be valid only when the sensor is in the enabled state.</p> <p>If the sensorOperationalState is set to enabled the sensor may temporarily return "Unknown" for the eventState if the sensor has not yet assessed a eventState value (as may happen immediately after the sensor has become enabled). Otherwise, the sensor must return a value other than "Unknown".</p> <p>The eventState value is set to "Unknown" when sensorOperationalState is set to any value except enabled. Parties that are using this command should ignore the eventState value under this condition. Refer to 17.6 for additional information about how presentState and eventState are generated.</p> <p>value: { Unknown, Normal, Warning, Critical, Fatal, LowerWarning, LowerCritical, LowerFatal, UpperWarning, UpperCritical, UpperFatal }</p>
uint8 sint8 uint16 sint16 sint32 uint32	<p>presentReading</p> <p>The present value indicated by the sensor</p> <p>NOTE The SensorDataSize field returns an enumeration that indicates the number of bits used to return the value. An implementation may either periodically sample the value and return the most recently collected sample, or it may sample the value at the time the presentReading is requested. The presentReading value is not required to return a correct value and must be ignored while the sensorOperationalState value of the sensor is Unavailable.</p>

1953 **18.3 GetSensorThresholds command**

1954 The GetSensorThresholds command is used to get the present threshold settings for a PLDM Numeric
 1955 Sensor. To avoid the need for a floating point format for threshold settings and the need for multibyte
 1956 multiplications and divisions in simple devices, the GetSensorThresholds and SetSensorThresholds
 1957 commands must use “raw” integers to be used in the conversion formula specified in the numeric sensor
 1958 PDR.

1959 Table 31 describes the format of the command.

1960 **Table 31 – GetSensorThresholds command format**

Type	Request data
uint16	sensorID A handle that is used to identify and access the sensor special values: 0x0000, 0xFFFF = reserved
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_SENSOR_ID = 0x80 }
enum8	sensorDataSize The bit width and format of reading and threshold values that the sensor returns value: { uint8, sint8, uint16, sint16, uint32, sint32 } NOTE The sensorDataSize return value provides an enumeration that indicates the number of bits used to return the threshold values. All six threshold fields must be returned regardless of which thresholds are implemented. If a given threshold is not implemented the implementation can elect to put any value in the corresponding field (0 is recommended). The Numeric Sensor PDRs describe which thresholds are supported and how the values are to be converted.
<i>For sensorDataSize = uint8 or sint8</i>	
uint8 sint8	upperThresholdWarning
uint8 sint8	upperThresholdCritical
uint8 sint8	upperThresholdFatal
uint8 sint8	lowerThresholdWarning
uint8 sint8	lowerThresholdCritical
uint8 sint8	lowerThresholdFatal
<i>For sensorDataSize = uint16 or sint16</i>	
uint16 sint16	upperThresholdWarning
uint16 sint16	upperThresholdCritical
uint16 sint16	upperThresholdFatal
uint16 sint16	lowerThresholdWarning
uint16 sint16	lowerThresholdCritical
uint16 sint16	lowerThresholdFatal

Type	Response data (continued)
<i>For sensorDataSize = uint32 or sint32</i>	
uint32 sint32	upperThresholdWarning
uint32 sint32	upperThresholdCritical
uint32 sint32	upperThresholdFatal
uint32 sint32	lowerThresholdWarning
uint32 sint32	lowerThresholdCritical
uint32 sint32	lowerThresholdFatal

1961 **18.4 SetSensorThresholds command**

1962 The SetSensorThresholds command is used to set the thresholds of a PLDM Numeric Sensor. Values for
 1963 all threshold parameters must be provided. However, if a particular threshold is not supported by the
 1964 sensor, the value passed in the corresponding parameter is ignored. The numeric sensor PDR indicates
 1965 which thresholds are supported. To avoid unintended event transitions, it is recommended that the sensor
 1966 be disabled while changing threshold settings. After disabling the sensor, it is recommended that a “read-
 1967 modify-write” operation be used to set the specific threshold values.

1968 Threshold values may be volatile or nonvolatile. The level of volatility is reflected in the PDR for the
 1969 sensor.

1970 To avoid the need for a floating point format for threshold settings and the need for multibyte
 1971 multiplications and divisions in simple devices, the GetSensorThresholds and SetSensorThresholds
 1972 commands must use “raw” integers to be used in the conversion formula specified in the numeric sensor
 1973 PDR.

1974 Table 32 describes the format of the command.

1975 **Table 32 – SetSensorThresholds command format**

Type	Request data
uint16	sensorID A handle that is used to identify and access the sensor special values: 0x0000, 0xFFFF = reserved
enum8	sensorDataSize The bit width and format for the thresholds that are set in the sensor value: { uint8, sint8, uint16, sint16, uint32, sint32 } NOTE This value is used for checking purposes only. A sensor accepts only one particular data format. The sensor data size must be known a priori; it can be obtained from a PDR for the sensor or by issuing a GetSensorThresholds command. Values for all six threshold parameters must be provided regardless of which thresholds are supported. If a particular threshold is not supported by the sensor, the value passed in the corresponding parameter is ignored.
<i>For sensorDataSize = uint8 or sint8</i>	
uint8 sint8	upperThresholdWarning
uint8 sint8	upperThresholdCritical
uint8 sint8	upperThresholdFatal
uint8 sint8	lowerThresholdWarning

Type	Request data (continued)
uint8 sint8	lowerThresholdCritical
uint8 sint8	lowerThresholdFatal
<i>For sensorDataSize = uint16 or sint16</i>	
uint16 sint16	upperThresholdWarning
uint16 sint16	upperThresholdCritical
uint16 sint16	upperThresholdFatal
uint16 sint16	lowerThresholdWarning
uint16 sint16	lowerThresholdCritical
uint16 sint16	lowerThresholdFatal
<i>For sensorDataSize = uint32 or sint32</i>	
uint32 sint32	upperThresholdWarning
uint32 sint32	upperThresholdCritical
uint32 sint32	upperThresholdFatal
uint32 sint32	lowerThresholdWarning
uint32 sint32	lowerThresholdCritical
uint32 sint32	lowerThresholdFatal
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80 }

1976 **18.5 RestoreSensorThresholds command**

1977 The RestoreSensorThresholds command restores default thresholds for the device. Table 33 describes
 1978 the format of the command.

1979 **Table 33 – RestoreSensorThresholds command format**

Type	Request data
uint16	sensorID A handle that is used to identify and access the sensor special values: 0x0000, 0xFFFF = reserved
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80 }

1980 **18.6 GetSensorHysteresis command**

1981 The GetSensorHysteresis command is used to read the present hysteresis setting for a PLDM Numeric
 1982 Sensor. The hysteresis value uses the same units, data size, and conversion factors that are specified for
 1983 the reading from the sensor. Table 34 describes the format of the command.

1984

Table 34 – GetSensorHysteresis command format

Type	Request data
uint16	sensorID A handle that is used to identify and access the sensor special values: 0x0000, 0xFFFF = reserved
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80 }
enum8	sensorDataSize The bit width of the hysteresis value that is being returned value: { uint8, sint8, uint16, sint16, uint32, sint32 }
<i>For sensorDataSize = uint8 or sint8</i>	
uint8 sint8	hysteresis value
<i>For sensorDataSize = uint16 or sint16</i>	
uint16 sint16	hysteresis value
<i>For sensorDataSize = uint32 or sint32</i>	
uint32 sint32	hysteresis value

1985

18.7 SetSensorHysteresis command

1986
1987
1988
1989

The SetSensorHysteresis command is used to set the present hysteresis setting for a PLDM Numeric Sensor. The hysteresis value uses the same units, data size, and conversion factors that are specified for the reading from the sensor. It is recommended that the sensor be disabled while changing the hysteresis setting. Table 35 describes the format of the command.

1990

Table 35 – SetSensorHysteresis command format

Type	Request data
uint16	<p>sensorID</p> <p>A handle that is used to identify and access the sensor</p> <p>special values: 0x0000, 0xFFFF = reserved</p>
enum8	<p>sensorDataSize</p> <p>The bit width and format for the following hysteresis value that is being set into the sensor value: { uint8, sint8, uint16, sint16, uint32, sint32 }</p> <p>NOTE This value is used for checking purposes only. A sensor accepts only one particular data format. The sensor data size must be known a priori; it can be obtained from a PDR for the sensor or by issuing a GetSensorHysteresis command.</p>
<i>For sensorDataSize = uint8 or sint8</i>	
uint8 sint8	hysteresis value
<i>For sensorDataSize = uint16 or sint16</i>	
uint16 sint16	hysteresis value
<i>For sensorDataSize = uint32 or sint32</i>	
uint32 sint32	hysteresis value
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80 }</p>

1991 **18.8 InitNumericSensor command**

1992 The InitNumericSensor command is typically used by the Initialization Agent function (see clause 15) to
 1993 initialize PLDM Numeric Sensors. The command may also be used as an interface for “virtual sensors,”
 1994 which do not actually poll and update their own state but instead rely on another management controller
 1995 or system software to set their state.

1996 Implementations should avoid virtual sensors that require initialization by the Initialization Agent function.
 1997 Conflicts could occur if the sensor needs to be accessed by the Initialization Agent function at the same
 1998 time it is being accessed as a virtual sensor. Typically, however, a virtual sensor would not require
 1999 initialization by the Initialization Agent function.

2000 Table 36 describes the format of the command.

2001

Table 36 – InitNumericSensor command format

Type	Request data
uint16	<p>sensorID</p> <p>A handle that is used to identify and access the sensor</p> <p>special values: 0x0000, 0xFFFF = reserved</p>
enum8	<p>sensorOperationalState</p> <p>The expected operational state of the sensor. This enumeration is a subset of the operational state values that are returned by the GetSensorReading command. Refer to the GetSensorReading command for the definition of the values in this enumeration.</p> <p>This parameter is applied to the sensor <i>after</i> all other fields (sensorPresentState, eventMsgEnable, and numericReadingSetting) have been applied to the sensor.</p> <p>value: { enabled, disabled, unavailable }</p>
enum8	<p>sensorPresentState</p> <p>The expected present state of the numeric sensor. See the description of the presentState field in Table 30.</p>
enum8	<p>eventMsgEnable</p> <p>This value is used to enable or disable event message generation from the sensor.</p> <p>value: {</p> <ul style="list-style-type: none"> enableEventMessages, disableEventMessages, noChange=0xFF // Do not alter the present event enable setting. <p>}</p>
bool8	<p>setNumericReading</p> <p>value: { false, true }</p> <p>True directs the receiver to accept the following numericReadingSetting.</p>
var	<p>numericReadingSetting</p> <p>The size of this field depends on the sensor data size. This value is used as the initial value for the presentReading returned by the numeric sensor. Some sensor implementations may ignore this value if it is given.</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80 }</p>

2002

19 PLDM State Sensors

2003

PLDM State Sensors are used to return a status from one or more state sets. A state set is simply the name of an enumeration that is a collection of a set of related platform states. Common state sets are defined in [DSP0249](#).

2004

2005

2006

A PLDM State Sensor that returns values from only a single state set is referred to as a simple state sensor. A state sensor that returns values from more than one state set is referred to as a composite state sensor.

2007

2008

2009 This specification also includes support for the definition of vendor-specific state sets using the OEM
 2010 State Set PDR. (See 28.10 for more information.)

2011 If a state sensor is reporting events or status and is based on a numeric sensor, the state sensor shall
 2012 use the threshold and hysteresis values for the associated numeric sensor for state change notification.
 2013 State Sensors that reflect logical states, such as redundancy, are device dependent and these sensor
 2014 types are outside the scope of this specification.

2015 **20 PLDM State Sensor commands**

2016 This clause describes the commands for accessing PLDM State Sensors per this specification. The
 2017 command numbers for the PLDM messages are given in clause 30.

2018 If PLDM State Sensors are implemented, the Mandatory/Conditional (M/C) requirements shown in Table
 2019 37 apply.

2020 **Table 37 – State Sensor commands**

Command	M/C	Reference
SetStateSensorEnables	M	See 20.1.
GetStateSensorReadings	M	See 20.2.
InitStateSensor	C ^[1]	See 20.3.

2021 ^[1] Required for sensors that are to be initialized through the Initialization Agent function.

2022 **20.1 SetStateSensorEnables command**

2023 The SetStateSensorEnables command is used to set enable or disable sensor operation and event
 2024 message generation for sensors within a PLDM Composite State Sensor. Event message generation is
 2025 optional for a sensor. Table 38 describes the format of the command.

2026 **Table 38 – SetStateSensorEnables command format**

Type	Request data
uint16	sensorID A handle that is used to identify and access the sensor special values: 0x0000, 0xFFFF = reserved
uint8	compositeSensorCount The number of individual sets of sensor information that this command accesses. Up to eight sets of state sensor information (accessed as sensor offsets 0 through 7) can be accessed through a given sensorID within a PLDM terminus. value: 0x01 to 0x08
opField xN	opFields Each opField is an instance of an opField structure that is used to set the present operational state setting and event message enables for a particular sensor within the state sensor. The opField structure is defined in Table 39.

Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80, EVENT_GENERATION_NOT_SUPPORTED = 0x82 }

2027

Table 39 – SetStateSensorEnables opField format

Type	Description
enum8	sensorOperationalState The expected state of the sensor This enumeration is a subset of the operational state values that are returned by the GetStateSensorReading command. Refer to the GetStateSensorReading command for the definition of the values in this enumeration. value: { enabled, disabled, unavailable }
enum8	eventMessageEnable This value is used to enable or disable event message generation from the sensor. value: { noChange, disableEvents, enableEvents, enableOpEventsOnly, enableStateEventsOnly } noChange means do not alter the present setting. Use noChange when the sensor does not support event message generation. NOTE Event message generation is optional for a sensor.

2028 **20.2 GetStateSensorReadings command**

2029 The GetStateSensorReadings command can return readings for multiple state sensors (a PLDM State
 2030 Sensor that returns more than one set of state information is called a composite state sensor).

2031 State information is returned as a sequence of one to N "stateField" structures. The first stateField
 2032 structure is referred to as the structure for the sensor at offset 0, second is for the sensor at offset 1, and
 2033 so on.

2034 The same number of stateField structures must be returned and in the same sequence during platform
 2035 management subsystem operation, regardless of the operational status of the sensors.

2036 Table 40 describes the format of the command.

2037

Table 40 – GetStateSensorReadings command format

Type	Request data
uint16	<p>sensorID</p> <p>A handle that is used to identify and access the simple or composite sensor special values: 0x00, 0xFFFF = reserved</p>
bitfield8	<p>sensorRearm</p> <p>Each bit location in this field corresponds to a particular sensor within the state sensor, where bit [0] corresponds to the first state sensor (sensor offset 0) and bit [7] corresponds to the eighth sensor (sensor offset 7), sequentially.</p> <p>For each bit position [n] from n = 0 to compositeSensorCount-1, the bit setting operates as follows:</p> <p>0b = do not re-arm sensor [n]+1 1b = re-arm sensor [n]+1</p>
uint8	<p>reserved</p> <p>value: 0x00</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80 }</p>
unit8	<p>compositeSensorCount</p> <p>The number of individual sets of sensor information that this command accesses. Up to eight sets of state sensor information (accessed as sensor offsets 0 through 7) can be accessed through a given sensorID within a PLDM terminus.</p> <p>value: 0x01 to 0x08</p>
stateField xN	<p>stateFields</p> <p>Each stateField is an instance of a stateField structure that is used to return the present operational state setting and the present state and event state for a particular set of sensor information contained within the state sensor. The stateField structure is defined in Table 41.</p>

2038

Table 41 – GetStateSensorReadings stateField format

Type	Description
enum8	<p>sensorOperationalState</p> <p>The state of the sensor itself See Table 30 for the enumeration values of sensorOperationalState.</p>
enum8	<p>presentState</p> <p>This field is used to return a state value from a PLDM State Set that is associated with the sensor. The value reflects the most recently assessed state.</p>

Type	Description
enum8	<p>previousState</p> <p>The state that the presentState was entered from. This must be different from the present state (with the exception that there may be conditions where both the presentState and previousState are returned as "Unknown").</p> <p>The previousState is updated whenever the presentState is assessed as different from the previously assessed value for presentState. Refer to 17.5 for additional information on how presentState is assessed.</p> <p>special value: This value shall be set to the same value as presentState if the previousState is unknown, which might be the case for events that are generated on the first status assessment that occurs after a sensor has been initialized.</p>
enum8	<p>eventState</p> <p>This field is used to return a state value from a PLDM State Set that is associated with the sensor. The value reflects the most recently assessed state that caused an event to be generated. The eventState can be different than either the presentState or the previousState.</p>

2039 **20.3 InitStateSensor command**

2040 The InitStateSensor command is typically used by the Initialization Agent function (see clause 15) to
 2041 initialize PLDM State Sensors. The command may also be used as an interface for virtual sensors, which
 2042 do not actually poll and update their own state but instead rely on another management controller or
 2043 system software to set their state.

2044 Implementations should avoid virtual sensors that require initialization by the Initialization Agent function.
 2045 Conflicts could occur if the sensor needs to be accessed by the Initialization Agent function at same time
 2046 it is being accessed as a virtual sensor. Typically, however, a virtual sensor would not require initialization
 2047 by the Initialization Agent function.

2048 Table 42 describes the format of the command.

2049 **Table 42 – InitStateSensor command format**

Type	Request data
uint16	<p>sensorID</p> <p>A handle that is used to identify and access the sensor</p> <p>special values: 0x0000, 0xFFFF = reserved</p>
unit8	<p>compositeSensorCount</p> <p>The number of individual sets of sensor information that this command accesses. Up to eight sets of state sensor information (accessed as sensor offsets 0 through 7) can be accessed through a given sensorID within a PLDM terminus.</p> <p>value: 0x01 to 0x08</p>
initField xN	<p>Each initField is an instance of an initField structure that is used to set the present operational state setting and event message enables for a particular sensor within the state sensor. The initField structure is defined in Table 43.</p>

Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, INVALID_SENSOR_ID = 0x80, UNSUPPORTED_SENSORSTATE = 0x81 // an illegal value was submitted for sensorOperationState or sensorPresentState for one or more sensors }</p>

2050

Table 43 – InitStateSensor initField format

Type	Description
enum8	<p>sensorOperationalState</p> <p>The expected operational state of the sensor. This enumeration is a subset of the operational state values that are returned by the GetSensorReading command. Refer to 18.2 for the definition of the values in this enumeration.</p> <p>This parameter is applied to the sensor after all other fields (sensorPresentState and eventMsgEnable) have been applied to the sensor.</p> <p>value: { enabled, disabled, unavailable }</p>
enum8	<p>sensorPresentState</p> <p>The expected state of the sensor. The state values are based on the particular state set used for the sensor. The set of states that the sensor can be initialized with may be a subset of the states that the sensor reports while monitoring.</p> <p>value: { dependent on sensor State Set }</p>
enum8	<p>eventMsgEnable</p> <p>This value is used to enable or disable event message generation from the sensor.</p> <p>value: { enableEvents, disableEvents, noChange=0xFF }</p> <p>noChange means do not alter the present setting.</p>

2051 **21 PLDM effecters**

2052 PLDM effecters provide a general mechanism for controlling or configuring a state or numeric setting of
 2053 an entity. PLDM effecters are similar to PLDM sensors, except that entity state and numeric setting values
 2054 are written into an effector rather than read from it.

2055 PLDM commands are specified for writing the state or numeric setting to an effector. Effecters are
 2056 identified by and accessed using an EffectorID that is unique for each effector within a given terminus.
 2057 Corresponding PDRs provide basic semantic information for effecters, such as what type of states or
 2058 numeric units the effector accepts, what terminus and EffectorID value are used to access the effector,
 2059 which entity the effector is associated with, and so on.

2060 **21.1 PLDM State Effecters**

2061 PLDM State Effecters provide a regular command structure for setting state information in order to
 2062 change the state of an entity. Effecters use the same PLDM State Sets definitions as PLDM State
 2063 Sensors, but instead of using the state set information to interpret the value that is read from a sensor,
 2064 the state sets are used to define the value to write to an effector. Like PLDM Composite State Sensors,
 2065 PLDM State Effecters can be implemented and accessed as composite state effecters where a single

2066 EffectorID is used to access a set of state effecters. This enables multiple states to be set using a single
 2067 command and to share a single PDR that provides the basic information for the effecters.

2068 **21.2 PLDM Numeric Effecters**

2069 PLDM Numeric Effecters provide a regular command structure for setting a numeric value for a
 2070 controllable parameter of an entity. Numeric effecters use the same definition of units as the units for
 2071 readings returned by numeric sensors (see 27.2). For example, a numeric effector could be used to set a
 2072 value for revolutions per second.

2073 **21.3 Effector semantics**

2074 An effector has a meaning or use that is associated with what an effector does or is used for. This will be
 2075 referred to as the "effector semantic", or just the "semantic."

2076 Although PLDM effecters provide a straightforward mechanism for setting a state or numeric value for an
 2077 entity, conveying the semantic of how that state or numeric value affects the entity, or how the setting
 2078 should be used, is not always straightforward.

2079 Suppose a numeric effector is defined for setting a fan speed. A PDR for the numeric effector can readily
 2080 indicate that the effector is for "Physical Fan 1", and that "Fan 1" is contained by Processor 1. The PDR
 2081 can also indicate that the units for the setting are "RPM". However, this does not convey what the RPM is
 2082 actually doing. For example, is the RPM a speed limit or a target speed?

2083 Additionally, other information may be necessary for understanding how the effector is to be used. If a fan
 2084 speed needs to be set because one or more temperatures have become too high, how does the user of
 2085 PLDM know which temperatures are associated with the fan, and what RPM value should be set for a
 2086 particular temperature?

2087 The information required to describe the meaning and use of an effector can vary significantly depending
 2088 on how generic or specific the use is to the platform implementation. The level of generality of effector
 2089 semantics in PLDM is categorized as shown in Table 44.

2090 **Table 44 – Categories for effector semantics**

Category	Description
By State Set or Units Only	The definition of the state set or numeric units, along with the Entity Association Information provided through the effector PDRs, is sufficient to convey the semantic for the effector. For example, the state set for System Power State when combined with "System" as the containerID identifies an effector for overall system power control.
By Semantic ID	The state sets or units definitions and entity associations alone are not sufficient to identify the semantic of the effector, but the effector use can be indicated by providing a single "Semantic ID" value that identifies a predefined semantic for the effector. For example, a Semantic ID could be defined for "System Power Down with Delay" where the definition specifies that the effector accepts a time value that identifies a delay from 1 to 60 seconds and triggers a system power down after that delay when the effector value gets set. This specification makes provision for DMTF PLDM defined or OEM (vendor-defined) Semantic IDs. See 21.4 for more information.
By Semantic ID plus PDRs	The effector PDR information and the Semantic ID are not sufficient to identify the semantic of the effector, but the semantic can be communicated when the Semantic ID is used with other PDRs. For example, an effector could be defined for setting a "Fan speed override" where the fan speed is set to a "boost mode" if one or more temperature sensors in the system exceed their critical thresholds. One or more additional PDRs would be used to identify which temperature sensors in the particular platform would contribute to boost mode. Note that in this case the effector itself is not implementing this policy. A third party, such as a MAP, would read the PDR information and use that information to know when it should change the effector's setting.

Category	Description
External Information Required	The effector semantic may not be described using the mechanisms offered by this specification. In some cases, use of the effector may require access to information that is not provided through PDRs—for example, an effector where the user (such as a MAP) requires access to SMBIOS data to understand how the effector should be used. In other cases, the effector semantic may have a private or proprietary where the effector is implemented using PLDM commands and described in the PDRs only because the implementation wants to reuse the command infrastructure from this specification or take advantage of functions such as the Initialization Agent or Event Log.

2091 The most generic and efficient use of effectors comes when they fall into the state sets or units only
 2092 category and use standard state set or units definitions. The second most generic and efficient use of
 2093 effectors is when they use a standard defined Semantic ID. Thus, if new standard effector semantics
 2094 need to be defined, it should be first examined whether a new state set or units definition should be
 2095 added to the specifications, or whether a new Semantic ID should be added.

2096 **21.4 PLDM and OEM effector semantic IDs**

2097 Effector Semantic ID values are specified in [DSP0249](#). A range of values is reserved for definition by the
 2098 DMTF PLDM specifications and another range of values is available for OEM (vendor-defined) effector
 2099 semantics. When the OEM range is used, the semantic is identified and optionally named using an OEM
 2100 Effector Semantic PDR. The use of the OEM Effector Semantic PDR is similar to how OEM units, entities,
 2101 and state sets are defined within the PDRs.

2102 **22 PLDM effector commands**

2103 This clause describes the commands for accessing PLDM effectors per this specification. The command
 2104 numbers for the PLDM messages are given in clause 30.

2105 If PLDM Numeric Effectors or PLDM State Effectors are implemented, the Mandatory (M) requirements
 2106 shown in Table 45 apply.

2107 **Table 45 – State and Numeric Effector commands**

Command	M	Reference
SetNumericEffectorEnable	M ^[1]	See 22.1.
SetNumericEffectorValue	M ^[1]	See 22.2.
GetNumericEffectorValue	M ^[1]	See 22.3.
SetStateEffectorEnables	M ^[2]	See 22.4.
SetStateEffectorStates	M ^[2]	See 22.5.
GetStateEffectorStates	M ^[2]	See 22.6.

2108 ^[1] Required if one or more numeric effectors are implemented

2109 ^[2] Required if one or more state effectors are implemented

2110 **22.1 SetNumericEffectorEnable command**

2111 The SetNumericEffectorEnable command is used to enable or disable effector operation. A disabled
 2112 effector cannot have its state updated. An effector may have a default state that it automatically returns to
 2113 when it is disabled. An effector may also be able to be returned to its default state through the
 2114 SetStateNumericEffectorValue command. The PLDM Numeric Effector PDR can describe a numeric
 2115 effector and whether it has a default state.

2116 NOTE The Numeric Effector PDR effectorID type, in clause 28.11 Numeric Effector PDR has been changed in
 2117 version 1.1.1 of this specification from uint8 to uint16 to be consistent with SetNumericEffectorEnable
 2118 command.

2119 Table 46 describes the format of this command.

2120 **Table 46 – SetNumericEffectorEnable command format**

Type	Request data
uint16	effectorID A handle that is used to identify and access the effector special values: 0x0000, 0xFFFF = reserved
enum8	effectorOperationalState The expected state of the effector. This enumeration is a subset of the operational state values that are returned by the GetStateEffectorStates command. Refer to the GetStateEffectorStates command for the definition of the values in this enumeration. value: { enabled, disabled = 2, unavailable }
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_EFFECTER_ID=0x80 }

2121 **22.2 SetNumericEffectorValue command**

2122 The SetNumericEffectorValue command is used to set the value for a PLDM Numeric Effector. Table 47
 2123 describes the format of this command.

2124 **Table 47 – SetNumericEffectorValue command format**

Type	Request data
uint16	effectorID A handle that is used to identify and access the effector special values: 0x0000, 0xFFFF = reserved
enum8	effectorDataSize The bit width and format of the setting value for the effector value: { uint8, sint8, uint16, sint16, uint32, sint32 } NOTE This value does not select a data size that is to be accepted by the effector. The value is used only to enable the responder to confirm that the effectorValue is being given in the expected format.
uint8 sint8 uint16 sint16 uint32 sint32	effectorValue The setting value of numeric effector being requested

Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_EFFECTER_ID=0x80, }

2125 **22.3 GetNumericEffectorValue command**

2126 The GetNumericEffectorValue command is used to return the present numeric setting of a PLDM Numeric
 2127 Effector. Table 48 describes the format of this command.

2128 **Table 48 – GetNumericEffectorValue command format**

Type	Request data
uint16	effectorID A handle that is used to identify and access the effector special values: 0x0000, 0xFFFF = reserved
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_EFFECTER_ID=0x80 }
enum8	effectorDataSize The bit width and format of the setting value for the effector value: { uint8, sint8, uint16, sint16, uint32, sint32 }

Type	Response data (continued)
enum8	<p>effectorOperationalState</p> <p>The state of the effector itself</p> <p>value: { enabled-updatePending, enabled-noUpdatePending, disabled, unavailable, statusUnknown, failed, initializing, shuttingDown, inTest }</p> <p>enabled-updatePending = Enabled and operating. The effector is able to return valid setting values. The setting of the numeric effector is in the process of being changed to the pending value.</p> <p>enabled-noUpdatePending = Enabled and operating. The effector is able to return valid setting values. The pending and presentValue fields return the present numeric setting of the effector.</p> <p>The pendingValue and presentValue fields may not be valid and should be ignored when the effector is in any of the following states. The implementation is not required to return any particular values for the pendingValue or presentValue fields in these states.</p> <p>disabled The effector is disabled from returning presentReading and event state values. This state is set through the SetNumericEffectorEnable command.</p> <p>unavailable The effector should be ignored due to configuration of the platform or monitored entity. For example, the effector is for monitoring a processor temperature, but the processor is not installed. This state is set through the SetNumericEffectorEnable command.</p> <p>statusUnknown The effector cannot presently return valid reading information for the monitored entity.</p> <p>failed The effector has failed. The effector implementation has determined that it cannot return correct values for its present setting.</p> <p>initializing The effector is in the process of transitioning to the operating state because the effector has been initialized (starting) or reinitialized. The presentState and eventState values shall be ignored while the effector is in this state.</p> <p>shuttingDown The effector is transitioning to the disabled, failed, or unavailable state.</p> <p>inTest The effector is presently undergoing testing.</p> <p>NOTE The operation of effector testing and the mechanisms for effector testing are outside the scope of this specification.</p>
uint8 sint8 uint16 sint16 sint32 uint32	<p>pendingValue</p> <p>The pending numeric value setting of the effector. The effectorDataSize field indicates the number of bits used for this field.</p>
uint8 sint8 uint16 sint16 sint32 uint32	<p>presentValue</p> <p>The present numeric value setting of the effector. The effectorDataSize indicates the number of bits used for this field.</p>

2129 **22.4 SetStateEffectorEnables command**

2130 The SetStateEffectorEnables command is used to enable or disable effector operation. A disabled
 2131 effector cannot have its state updated. An effector may have a default state that it automatically returns to
 2132 when it is disabled. An effector may also be able to be returned to its default state through the

2133 SetStateEffectorStates command. The PLDM State Effector PDR describes a state effector and whether
 2134 it has a default state. Table 49 describes the format of this command.

2135 **Table 49 – SetStateEffectorEnables command format**

Type	Request data
uint16	effectorID A handle that is used to identify and access the effector special values: 0x0000, 0xFFFF = reserved
uint8	compositeEffectorCount The number of individual sets of state effector information that are accessed by this command. Up to eight sets of effector information (accessed as effector offsets 0 through 7) can be accessed through a given effectorID within a PLDM terminus. value: 0x01 to 0x08
opField xN	opFields Each opField is an instance of an opField structure that is used to set the present operational state setting and event message enables for a particular sensor within the state effector. The opField structure is defined in Table 50.
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_EFFECTER_ID=0x80 }

2136 **Table 50 – SetStateEffectorEnables opField format**

Type	Description
enum8	effectorOperationalState The expected state of the effector. This enumeration is a subset of the operational state values that are returned by the GetStateEffectorStates command. Refer to the GetStateEffectorStates command for the definition of the values in this enumeration. value: { enabled, disabled=2, unavailable }
enum8	eventMsgEnable This value is used to enable or disable event message generation from the effector. value: { enableEvents, disableEvents, noChange=0xFF } noChange means do not alter the present setting.

2137 **22.5 SetStateEffectorStates command**

2138 The SetStateEffectorStates command is used to set the state of one or more effecters within a PLDM
 2139 State Effector. Table 51 describes the format of this command.

2140 **Table 51 – SetStateEffectorStates command format**

Type	Request data
uint16	effectorID A handle that is used to identify and access the effector special values: 0x0000, 0xFFFF = reserved
unit8	compositeEffectorCount The number of individual sets of effector information that are accessed by this command. Up to eight sets of state effector information (accessed as effector offsets 0 through 7) can be accessed through a given effectorID within a PLDM terminus. value: 0x01 to 0x08
stateField xN	Each stateField is an instance of a stateField structure that is used to set the requested state for a particular effector within the state effector. The stateField structure is defined in Table 52.
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_EFFECTER_ID=0x80, INVALID_STATE_VALUE=0x81, UNSUPPORTED_EFFECTERSTATE = 0x82 // An illegal value was submitted for effectorState for one or more effecters. }

2141 **Table 52 – SetStateEffectorStates stateField format**

Type	Description
enum8	setRequest value: { noChange, // Do not request a change of the state of this effector. requestSet // Request the effector state to be set to the state given by the following // effectorState value. }
enum8	effectorState The expected state of the effector. The state values come from the particular state set used for the implementation of the effector. value: { dependent on effector state set }

2142 **22.6 GetStateEffectorStates command**

2143 The GetStateEffectorStates command is used to get the present state of an effector. Table 53 describes
 2144 the format of this command.

2145 **Table 53 – GetStateEffectorStates command format**

Type	Request data
uint16	effectorID A handle that is used to identify and access the simple or composite effector special values: 0x0000, 0xFFFF = reserved
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_EFFECTER_ID=0x80 }
unit8	compositeEffectorCount The number of individual sets of effector information that are accessed by this command. Up to eight sets of state effector information (accessed as effector offsets 0 through 7) can be accessed through a given effectorID within a PLDM terminus. value: 0x01 to 0x08
stateField xN	stateFields Each stateField is an instance of a stateField structure that is used to return the present operational state setting and the present state for a particular effector contained within the state effector. The stateField structure is defined in Table 54.

2146 **Table 54 – GetStateEffectorStates stateField format**

Type	Description
enum8	effectorOperationalState The state of the effector itself See Table 48 for the enumeration values of effectorOperationalState.
enum8	pendingState If the value of effectorOperationalState is updatePending, this field returns the value for the requested state that is presently being processed. Otherwise, this field returns the present state of the effector. The effector implementation should return the "Unknown" state value whenever the effectorOperationalState is anything except enabled-updatePending or enabled-noUpdatePending. Parties that are accessing this information should also ignore this field (treat it as unknown) when the effectorOperationalState is anything except enabled-updatePending or enabled-noUpdatePending. value: { dependent on effector state set on which the effector implementation is based }
enum8	presentState The present state of the effector. The effector implementation should return the "Unknown" state value whenever the value of effectorOperationalState is anything except enabled-updatePending or enabled-noUpdatePending. Parties that are accessing this information should also ignore this field (treat it as unknown) when the effectorOperationalState is anything except enabled-updatePending or enabled-noUpdatePending. value: { dependent on the state set used for the effector implementation }

2147 **23 PLDM Event Log commands**

2148 This clause describes the commands for accessing a PLDM Event Log per this specification. The
 2149 command numbers for the PLDM messages are given in clause 30.

2150 The PLDM Event Log is typically accessed through the same PLDM terminus as the Event Receiver.
 2151 However, this is not mandatory. The PDRs include information that describes which terminus is used to
 2152 access the PLDM Event Log.

2153 If a PLDM Event Log is implemented, the Mandatory/Optional/Conditional (M/O/C) requirements shown in
 2154 Table 55 apply.

2155 **Table 55 – PLDM Event Log commands**

Command	M/O/C	Reference
GetPLDMEventLogInfo	M	See 23.1.
EnablePLDMEventLogging	M	See 23.2.
ClearPLDMEventLog	M	See 23.3.
GetPLDMEventLogTimestamp	M	See 23.4.
SetPLDMEventLogTimestamp	M	See 23.5.
ReadPLDMEventLog	M	See 23.6.
GetPLDMEventLogPolicyInfo	M	See 23.7.
SetPLDMEventLogPolicy	C ^[1]	See 23.8.
FindPLDMEventLogEntry	O	See 23.9

2156 ^[1] Required if the PLDMEventLog implementation supports configurable policy parameters

2157 **23.1 GetPLDMEventLogInfo command**

2158 The GetPLDMEventLogInfo command returns basic information about the PLDM Event Log, such as its
 2159 operational status, percentage used, and timestamps for the most recent add and erase actions. Table 56
 2160 describes the format of the command.

2161 **Table 56 – GetPLDMEventLogInfo command format**

Type	Request data
–	none
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
enum8	logOperationalStatus value: { loggingDisabled, // Log can be accessed, but is disabled from accepting entries. enabledReady, // Log can be accessed and is enabled to accept entries. clearInProgress, // Log is enabled but log information and entries are unable to be // accessed because the log is in the process of being cleared. enabledFull, // Log is enabled but cannot accept more entries because it is // full. The log shall automatically resume accepting entries once // entries are cleared. It is not necessary to explicitly re-enable // logging. failedLoggingDisabled, // Log has had a failure where it can no longer accept entries. // Clearing and re-enabling logging must restore the log to // normal operation. If this cannot occur, the 'failedDisabled' // logOperationalStatus value shall be returned. failedDisabled, // Log has had a failure where it is unable to // accept entries. Additionally, existing entries may not be able // to be accessed successfully. The log may or may not be able // to be restored to normal operation by clearing and re-enabling // the log. corrupted // Some or all log data has been lost due to a data corruption. // Clearing the log and re-enabling logging shall restore internal // integrity. If this cannot be done, the implementation shall // return a logOperationalStatus of failedLoggingDisabled or // failedDisabled. The log implementation shall not return records // that are known to be corrupted. }
enum8	activeLogClearingPolicy The log clearing policy that is presently in effect for this PLDM Event Log. See 13.4 for a description of the log clearing policies. value: { fillAndStop, FIFO, clearOnAge }

Type	Response data (continued)
uint32	<p>entryCount</p> <p>number of entries presently in the Event Log</p>
uint8	<p>storagePercentUsed</p> <p>The percentage of log storage space presently used up by entries in the log, given in increments based on the percentUsedResolution parameter from the PLDM Event Log PDR</p> <p>value: 0 to 100</p> <p>special value: 0xFF = unspecified</p>
uint8	<p>percentWear</p> <p>The implementation may elect to return this value as an indication of the present level of wear on the storage medium. Values 0 to 100 indicate an estimated percentage of normal rated lifetime or storage cycles used up on the device. Values greater than 100 indicate levels that have exceeded the rated or expected lifetime. The mechanism and algorithms that are used for returning this parameter are implementation-specific and outside the scope of this specification.</p> <p>value: 0x00 to 0x064 = wear in %</p> <p>special value: 0xFF = unspecified</p>
<p>mostRecentAddTimestamp</p> <p>The following three fields return the timestamp of the most recent addition or change to the log.</p> <p>The implementation must automatically adjust the mostRecentAddTimestamp whenever the Event Log timestamp clock is set using the SetPLDMEventLogTimestamp command. See the description of the SetPLDMEventLogTimestamp command for more information.</p> <p>special value: The implementation may choose to retain the mostRecentAddTimestamp value after the log has been cleared, or it may elect to set the value to the 'unspecified' value for the data type. The unspecified value shall only be used when the log is empty (cleared), or if the timestamp has been lost due to an error or firmware update condition.</p>	
sint8	<p>mostRecentAddTimestampUTCOffset</p> <p>The UTC offset for the log entry timestamp in increments of 1/2 hour.</p> <p>special value: 0xFF = unspecified</p>
uint40	<p>mostRecentAddTimestampSeconds</p> <p>This value corresponds to a 40-bit unsigned integer representing the number of seconds since midnight UTC of January 1, 1970 (not counting leap seconds). 0x0000000000 = unspecified.</p>
uint8	<p>mostRecentAddTimestamp100s</p> <p>This value provides a number of 1/100ths of a second added to entryTimestampSeconds.</p> <p>value: 0 to 99.</p> <p>special value: 0xFF = unspecified. This value is used if the implementation timestamps entries to no finer than a one-second resolution.</p>
<p>mostRecentEraseTimestamp</p> <p>The following three fields return the most recent time that entries were deleted from the log or the log was cleared.</p> <p>The implementation must automatically adjust the mostRecentEraseTimestamp whenever the Event Log timestamp clock is set using the SetPLDMEventLogTimestamp command. See the description of the SetPLDMEventLogTimestamp command for more information.</p> <p>special value: The implementation may choose to retain the mostRecentAddTimestamp value after the log has been cleared, or it may elect to set the value to the 'unspecified' value for the data type. The unspecified value shall only be used if the timestamp has never been initialized, or if the timestamp has been lost due to an error or firmware update condition.</p>	

Type	Response data (continued)
sint8	<p>mostRecentEraseTimestampUTCOffset</p> <p>The UTC offset for the log entry timestamp in increments of 1/2 hour. special value: 0xFF = unspecified</p>
uint40	<p>mostRecentEraseTimestampSeconds</p> <p>This value corresponds to a 40-bit unsigned integer representing the number of seconds since midnight UTC of January 1, 1970 (not counting leap seconds). 0x0000000000 = unspecified.</p>
uint8	<p>mostRecentEraseTimestamp100s</p> <p>This value provides a number of 1/100ths of a second added to entryTimestampSeconds. value: 0 to 99. special value: 0xFF = unspecified. This value is used if the implementation timestamps entries to no finer than a one-second resolution.</p>

2162 **23.2 EnablePLDMEventLogging command**

2163 The EnablePLDMEventLogging command is used to enable or disable the PLDM Event log from logging
 2164 events. The log can be accessed and cleared while in the disabled state unless the logOperationalStatus
 2165 is "failed", in which case logging may not be able to be enabled. Table 57 describes the format of the
 2166 command.

2167 **Table 57 – EnablePLDMEventLogging command format**

Type	Request data
enum8	<p>enableLogging</p> <p>value: { disableLogging, // Disable accepting events into the log. enableLogging // Enable logging events. } </p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES }</p>
enum8	<p>logOperationalStatus</p> <p>value: { See the definition of logOperationalStatus field for the GetPLDMEventLogInfo command (Table 56). }</p>

2168 **23.3 ClearPLDMEventLog command**

2169 The ClearPLDMEventLog command is used to clear the contents of the PLDM Event Log. The execution
 2170 of this command does not affect whether logging is enabled or disabled. Depending on the subsystem
 2171 and its implementation, it is possible that events may be received or be in the process of being received
 2172 during the terminus' execution of this command. If event logging is enabled, a terminus should continue to
 2173 accept events while it is processing this command. It is recognized that in some implementations clearing
 2174 the log device may take a significant amount of time. The number of events that an implementation may
 2175 support queuing up while the log is being cleared is implementation dependent. Table 58 describes the
 2176 format of this command.

2177

Table 58 – ClearPLDMEventLog command format

Type	Request data
–	none
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
enum8	logOperationalStatus The status of the log following acceptance of this command. This status will typically be clearInProgress, enabledReady, or loggingDisabled, depending on the implementation. value: { See the definition of logOperationalStatus for the GetPLDMEventLogInfo command (Table 59). }

2178 **23.4 GetPLDMEventLogTimestamp command**

2179 The GetPLDMEventLogTimestamp command returns a snapshot of the present PLDM Event Log
2180 Timestamp time. Table 59 describes the format of this command.

2181 **Table 59 – GetPLDMEventLogTimestamp command format**

Type	Request data
–	none
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
sint8	entryTimestampUTCOffset The UTC offset for the log entry timestamp in increments of 1/2 hour special value: 0xFF = unspecified
uint40	entryTimestampSeconds This value corresponds to a 40-bit unsigned integer that represents the number of seconds since midnight UTC of January 1, 1970 (not counting leap seconds).
uint8	entryTimestamp100s This value provides a number of 1/100 of a second that is added to entryTimestampSeconds . value: 0 to 99 special value: 0xFF = unspecified. This value is used if the implementation timestamps entries to no finer than a one-second resolution.

2182 **23.5 SetPLDMEventLogTimestamp command**

2183 The SetPLDMEventLogTimestamp command can be used to set the PLDM Event Log Timestamp time.

2184 Some implementations may not implement the ability to set the timestamp to 1/100 of a second resolution
 2185 and will round the time up or down to match the resolution that it supports. Therefore, the timestamp
 2186 value in the response may vary from what was submitted because of rounding. The returned value may
 2187 also vary due to delays in command response processing within the terminus.

2188 Implementations are required to support a 1 second or finer resolution for the timestamp. Table 60
 2189 describes the format of this command.

2190 **Table 60 – SetPLDMEventLogTimestamp command format**

Type	Request data
sint8	<p>entryTimestampUTCOffset</p> <p>The UTC offset for the log entry timestamp in increments of 1/2 hour special value: 0xFF = unspecified</p>
uint40	<p>entryTimestampSeconds</p> <p>This value corresponds to a 40-bit unsigned integer that represents the number of seconds since midnight UTC of January 1, 1970 (not counting leap seconds).</p>
uint8	<p>entryTimestamp100s</p> <p>This value provides a number of 1/100 of a second that is added to entryTimestampSeconds. value: 0 to 99</p> <p>This value is ignored if the implementation only timestamps entries to a one-second resolution.</p>
enum8	<p>logUpdateEvent</p> <p>value: {</p> <ul style="list-style-type: none"> noEvent, logEvent // automatically logs a timestamp change event if the new timestamp clock // value is accepted. See DSP0249 for the state set definition for time // stamp change events. <p>}</p>

2191

Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
sint8	entryTimestampUTCOffset The UTC offset for the log entry timestamp in increments of 1/2 hour special value: 0xFF = unspecified
uint40	entryTimestampSeconds This value corresponds to a 40-bit unsigned integer that represents the number of seconds since midnight UTC of January 1, 1970 (not counting leap seconds).
uint8	entryTimestamp100s This value provides a number of 1/100 of a second that is added to entryTimestampSeconds . value: 0 to 99 special value: 0xFF = unspecified. This value is used if the implementation timestamps entries to no finer than a one-second resolution.
uint8	timestampResolution The resolution of the timestamp that is kept by the implementation in 1/100 of a second. value: 1 to 100 (100 = 1 second resolution, 5 = .05 seconds resolution, and so on)

2192 **23.6 ReadPLDMEventLog command**

2193 The ReadPLDMEventLog command can be used iteratively to read all or part of the entries in the PLDM
2194 Event Log. Entries are returned one at a time. The data for one or more entries may be requested. Table
2195 61 describes the format of this command.

2196 To use the command to start reading from the first entry in the log:

- 2197 • Set entryID to 0 and transferOperationFlag to GetFirstPart.
- 2198 • Issue the command to get the first portion of data for the first entry in the log.
- 2199 • Take the nextEntryID and nextTransferOperationFlag data from the response and use it as the
2200 entryID and transferOperationFlag for the next request.
- 2201 • Repeat this until the desired number of entries have been read or the end of the log has been
2202 reached.

2203 The FindPLDMEventLogEntry command can be used to get the entryID for an entry that is at an offset
2204 into the log, or that has a timestamp that is older or newer than a given value. This entryID can then be
2205 used in the ReadPLDMEventLog command, along with setting transferOperationFlag = GetFirstPart, to
2206 begin reading the log starting with the found entry.

Table 61 – ReadPLDMEventLog command format

Type	Request data
uint32	<p>entryID</p> <p>A handle that identifies a particular log entry to be transferred or that is in the process of being transferred. The entryID values for the first portion of a given record are required to be unique and unchanging among all entries that are presently in the log. If the data for the entry is split across multiple responses, the entryID is also used to track which portion of the record is being returned in the response. How this is accomplished is implementation specific. For example, one possible implementation would be to use the upper bits of the entryID as an ID for the overall record, and the least significant bits of entryID to track an offset into the record.</p> <p>The entryID that is delivered in the response when in the middle of a multipart transfer (splitEntry = firstFragment or middleFragment) is allowed to time out. The timeout value is specified in the Event Log PDR. This provision is made to allow the responder implementation to assign a temporary ID and buffer space that can be freed up if the requester does not complete the multipart transfer of an entry. The default value for the timeout is the same value that is used for PDR Handle Timeouts, MC1. (See clause 28.25.) If PDRs are not used, a requester should assume the default timeout value is being used unless the requester has a priori knowledge of the implementation.</p> <p>value: Set to 0x00000000 and transferOperationFlag = GetFirstPart to start reading from the first (oldest) entry in the log;</p>
enum8	<p>transferOperationFlag</p> <p>The operation flag indicates whether this is the start of a new transfer or the continuation of a multipart transfer of an entry. GetFirstPart identifies transfer of the first entry of a multiple entry read. GetNextPart refers to a request to transfer entries that follow the first entry in a multiple entry transfer.</p> <p>Possible values: {GetNextPart=0x00, GetFirstPart=0x01}</p>
Type	Response data
enum8	<p>completionCode</p> <p>Possible values:</p> <p>{ PLDM_BASE_CODES, INVALID_TRANSFER_OPERATION_FLAG=0x81, INVALID_ENTRY_ID=0x82, }</p>
uint32	<p>nextEntryID</p> <p>An implementation-specific handle that is used by the implementation to track and identify the next portion of the transfer. This value is used as the dataTransferHandle to retrieve the next portion of eventLog data. Note that if the value for the splitEntry field (below) is firstFragment or middleFragment, the nextEntryID value is an ID that identifies the next <i>portion</i> of the record that is being transferred. If splitEntry field is full or lastFragment, the nextEntryID is the ID for the first portion of the next record in the log.</p> <p>special value: 0x00000000 = No next record. This value is only allowed when splitEntry = full or lastFragment. It indicates that there are no records that follow in the log. That is, the PLDMEventLogData that is being returned in the response holds the last portion of data for the last record in the log.</p>

Type	Response data (continued)
enum8	<p>splitEntry</p> <p>value: {</p> <p>full, // All of the data for the entry is provided in the entryData field.</p> <p>firstFragment, // The eventData for the entry is split across ReadPLDMEventLog messages. // The entryData field holds the first portion of the data for the entry.</p> <p>middleFragment, // The eventData for the entry is split across ReadPLDMEventLog messages. // The entryData field holds a middle portion of the data for the entry.</p> <p>lastFragment // The eventData for the entry is split across ReadPLDMEventLog messages. // The entryData field holds the last portion of the data for the entry.</p> <p>}</p>
–	<p>PLDMEventLogData</p> <p>The data or partial data for the requested PLDM Event Log entry. Entries are transferred starting from the oldest to the newest.</p>
<i>If splitEntry = lastFragment</i>	
uint8	<p>transferCRC</p> <p>A CRC-8 for the overall PLDM Event Log entry. This is provided to help verify data integrity when the entry is transferred using a multipart transfer. The CRC is calculated over the entire PLDM Event Log entry data as specified in Table 6 using the polynomial $x^8 + x^2 + x^1 + 1$ (This is the same polynomial used in the MCTP over SMBus/I²C transport binding specification). The CRC is calculated from most-significant bit to least-significant bit on bytes in the order that they are received. This field is only present when splitEntry = lastFragment.</p>

2208

Table 62 – PLDMEventLogData format

Type	Field
uint8	<p>transferredDataSize</p> <p>If splitEntry = full, then dataSize = number of bytes of entryData for the entire entry.</p> <p>If splitEntry = firstFragment, middleFragment, or lastFragment, then dataSize = number of bytes of entryData for the portion that is being transferred.</p>
–	<p>transferredEntryData</p> <p>Data for all or part of an event log entry, depending on whether the entry is split across PLDM messages. See 13.7 for PLDM Event Log entry formats.</p>

2209 **23.7 GetPLDMEventLogPolicyInfo command**

2210 The GetPLDMEventLogPolicyInfo command returns details about the different log clearing policies that
 2211 are supported for the particular PLDM Event Log implementation. Table 63 describes the format of this
 2212 command.

2213

Table 63 – GetPLDMEventLogPolicyInfo command format

Type	Request data
enum8	<p>logClearingPolicy</p> <p>This parameter selects the logClearingPolicy for which information is to be returned. See 13.4 for a description of the log clearing policies. The command returns the same fields regardless of whether they are used by the selected policy. Fields are filled with a special value if they are not used by the policy. The PLDM Event Log PDR indicates which policies are supported.</p> <p>value: { fillAndStop, FIFO, clearOnAge }</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES }</p>
bitfield8	<p>configurableParameterSupport</p> <p>This information and the following fields are specific to the logClearingPolicy that was selected in the request.</p> <p>[7:5] – reserved</p> <p>[4:3] – 00b = M and MPercentage are not configurable. 01b = M is configurable 10b = MPercentage is configurable. 11b = reserved</p> <p>[2:1] – 00b = N and NPercentage are not configurable. 01b = N is configurable. 10b = NPercentage is configurable. 11b = reserved</p> <p>[0] – 1b = Age is configurable.</p>
uint32	<p>NMin</p> <p>The smallest number that the implementation accepts or uses as a value for N for the given logClearingPolicy (see 13.4).</p> <p>special value: Return 0x00000000 if the policy implementation uses NPercentage instead of N, or if the policy does not use an N value.</p>
uint32	<p>NMax</p> <p>The largest number that the implementation accepts or uses as a value for N for the given logClearingPolicy (see 13.4).</p> <p>special value: Return 0x00000000 if the policy implementation uses NPercentage instead of N, or if the policy does not use an N value.</p>

Type	Response data (continued)
uint8	<p>NPercentageMin</p> <p>The smallest number that the implementation accepts or uses as a value for NPercentage for the given logClearingPolicy (see 13.4).</p> <p>value: 1 to 100; all other values = reserved</p> <p>special value: Return 0x00 if the policy implementation uses N instead of NPercentage, or if the policy does not use an NPercentage value.</p>
uint8	<p>NPercentageMax</p> <p>The largest number that the implementation accepts or uses as a value for NPercentage for the given logClearingPolicy (see 13.4).</p> <p>value: 1 to 100; all other values = reserved</p> <p>special value: Return 0x00 if the policy implementation uses N instead of NPercentage, or if the policy does not use an NPercentage value.</p>
uint32	<p>MMin</p> <p>The smallest number that the implementation accepts or uses as a value for M for the given logClearingPolicy (see 13.4).</p> <p>special value: Return 0x00000000 if the policy implementation uses MPercentage instead of M, or if the policy does not use an M value.</p>
uint32	<p>MMax</p> <p>The largest number that the implementation accepts or uses as a value for M for the given logClearingPolicy (see 13.4).</p> <p>special value: Return 0x00000000 if the policy implementation uses MPercentage instead of M, or if the policy does not use an M value.</p>
uint8	<p>MPercentageMin</p> <p>The smallest number that the implementation accepts or uses as a value for MPercentage for the given logClearingPolicy (see 13.4).</p> <p>value: 1 to 100; all other values = reserved</p> <p>special value: Return 0x00 if the policy implementation uses M instead of MPercentage, or if the policy does not use an MPercentage value.</p>
uint8	<p>MPercentageMax</p> <p>The largest number that the implementation accepts or uses as a value for MPercentage for the given logClearingPolicy (see 13.4).</p> <p>value: 1 to 100; all other values = reserved</p> <p>special value: Return 0x00 if the policy implementation uses M instead of MPercentage, or if the policy does not use an MPercentage value.</p>
uint32	<p>ageMin</p> <p>The smallest value that the implementation accepts or uses as a value for age in seconds for the given logClearingPolicy (see 13.4).</p> <p>special value: Return 0x00000000 if the policy does not use an age value.</p>
uint32	<p>ageMax</p> <p>The largest value that the implementation accepts or uses as a value for age in seconds for the given logClearingPolicy (see 13.4).</p> <p>special value: Return 0x00000000 if the policy does not use an age value.</p>

2214 **23.8 SetPLDMEventLogPolicy command**

2215 The SetPLDMEventLogPolicy command is used to select and configure the PLDM Event Log clearing
 2216 policies. Table 64 describes the format of the command.

2217 **Table 64 – SetPLDMEventLogPolicy command format**

Type	Request data
enum8	<p>selectedLogClearingPolicy</p> <p>This parameter selects the log clearing policy to be used by the PLDM Event Log. See 13.4 for a description of the log clearing policies.</p> <p>value: { fillAndStop, FIFO, clearOnAge }</p>
enum8	<p>setOperation</p> <p>value: {</p> <p>configureOnly, // Change the configuration of the policy identified by // selectedLogClearingPolicy by using the following configuration parameters, // but do not change which policy is selected as the active policy.</p> <p>setOnly, // Set the active policy to the policy identified by selectedLogClearingPolicy, but // do not set any of the configuration parameters. If this setOperation is used, // the following configuration parameters in the request shall be ignored by the // responder.</p> <p>configureAndSet // Set the active policy to the policy identified by selectedLogClearingPolicy and // set the configuration parameters for the selected policy using the following // configuration parameters.</p> <p>}</p>
uint32	<p>N</p> <p>The number of entries that will be automatically cleared for the given selectedLogClearingPolicy. See 13.4 for a description of the log clearing policies.</p> <p>special value: Use 0x00000000 if the policy implementation does not support a configurable N value. If the responder does not support a configurable N value, an error completionCode must be returned if this is set to a value other than 0.</p>
uint8	<p>NPercentage</p> <p>The percentage of the log that will be automatically cleared for the given selectedLogClearingPolicy. See 13.4 for a description of the log clearing policies.</p> <p>value: 1 to 100; all other values = reserved</p> <p>special value: Use 0x00 if the policy implementation does not support NPercentage as a configurable value. If the responder does not support a configurable NPercentage value, an error completionCode must be returned if this is set to a value other than 0.</p>
uint32	<p>M</p> <p>The number of entries that must be in the log before entries will be automatically cleared based on the selectedLogClearingPolicy. See 13.4 for a description of the log clearing policies.</p> <p>special value: Use 0x00000000 if the policy implementation does not support a configurable M value. If the responder does not support a configurable M value, an error completionCode must be returned if this is set to a value other than 0.</p>

Type	Request data (continued)
uint8	<p>MPercentage</p> <p>The percentage of the log that must be filled before entries will be automatically cleared based on the selectedLogClearingPolicy. See 13.4 for a description of the log clearing policies.</p> <p>value: 1 to 100; all other values = reserved</p> <p>special value: Use 0x00 if the policy does not support MPercentage as a configurable value. If the responder does not support a configurable MPercentage value, an error completionCode must be returned if this is set to a value other than 0.</p>
uint32	<p>age</p> <p>This parameter sets the age interval in seconds for the given selectedLogClearingPolicy. See 13.4 for a description of the log clearing policies.</p> <p>special value: Use 0x00000000 if the policy implementation does not support a configurable age. If the responder does not support a configurable age, an error completionCode must be returned if this is set to a value other than 0.</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES }</p>

2218 **23.9 FindPLDMEventLogEntry command**

2219 This command can be used to obtain the Entry ID value for the first entry in the Event Log that meets the
 2220 identified search parameter. This value can then be used in the ReadPLDMEventLog command to start
 2221 reading the log from that entry onward. The search parameters support finding the first entry that is newer
 2222 or older than a specified timestamp value, or the entry that corresponds to a particular offset from the
 2223 start or the present end of the log. Table 65 describes the format of this command.

2224 NOTE The order of fields in the response message for this command has been changed to having the
 2225 completionCode before the entryID in version 1.2.0 of this specification; this achieves consistency with all
 2226 other PLDM commands.

2227

Table 65 – FindPLDMEventLogEntry command format

Type	Request data
enum8	<p>searchType</p> <p>value: {newerThan, olderThan, offsetFromStart, offsetFromEnd}</p>
uint32	<p>startingPoint</p> <p>The EntryID for the log entry or the offset from which searching will start. Searches include the entry at the identified starting point.</p> <p>The search always occurs in the direction from the start of the log (first entries) to the end of the log (last entries).</p> <p>If searchType = newerThan or olderThan:</p> <p style="padding-left: 40px;">A nonzero value indicates an EntryID to start searching from. Use the value 0x00000000 to start searching from the first entry in the log. Use the value 0xFFFFFFFF to start searching from the last entry in the log.</p> <p>If searchType = offsetFromStart:</p> <p style="padding-left: 40px;">The value identifies the Nth entry from the start of the log. For example, if starting point = 10 the search will start with the 10th entry at the beginning of the log. An error completionCode shall be returned if the value exceeds the number of entries in the log.</p> <p>If searchType = offsetFromEnd:</p> <p style="padding-left: 40px;">The value identifies the Nth entry from the end of the log. For example, if starting point = 10 and the log contains 100 entries, the search will start with the 91st entry. An error completionCode shall be returned if the value exceeds the number of entries in the log.</p>
<p>compareTimestamp</p> <p><i>The compareTimestamp fields are only present when searchType = newerThan or olderThan.</i></p> <p><i>If searchType = newerThan, the response will hold the entryID for the first log entry that was found with a timestamp that is more recent than or equal to compareTimestamp.</i></p> <p><i>If searchType = olderThan, the response will hold the entryID for the first log entry that was found with a timestamp that is older than or equal to compareTimestamp.</i></p>	
sint8	<p>compareTimestampUTCOffset</p> <p>The UTC offset for the log entry timestamp in increments of 1/2 hour.</p> <p>special value: 0xFF = unspecified</p>
uint40	<p>compareTimestampSeconds</p> <p>This value corresponds to a 40-bit unsigned integer representing the number of seconds since midnight UTC of January 1, 1970 (not counting leap seconds). 0x0000000000 = unspecified.</p>
uint8	<p>compareTimestamp100s</p> <p>This value provides a number of 1/100ths of a second added to entryTimestampSeconds.</p> <p>value: 0 to 99.</p> <p>special value: 0xFF = unspecified. This value is used if the implementation timestamps entries to no finer than a one-second resolution.</p>

Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_SEARCH_TYPE = 0x80 }
uint32	entryID The entryID for the found log entry. This value can be used in the ReadPLDMEventLog command. special value: 0xFFFFFFFF = Not found. The command did not find a record matching the searchType.

2228 24 PLDM State Sets

2229 PLDM State Sets are specified enumerations for sets of state information that can be returned from
 2230 PLDM state sensors. State sets may also be used to provide a common definition for state information
 2231 used by other parts of PLDM.

2232 The state sets are the basis of state data that can be mapped as a data source into CIM properties that
 2233 return state information, and also provide state information that can be used for monitoring and controlling
 2234 the operation of PLDM itself.

2235 PLDM State Sets are defined in [DSP0249](#). This specification defines a numeric ID for each different state
 2236 set, defines the enumeration values for the states that make up the set, and provides definitions for each
 2237 state within the set. Because the state sets are expected to be extended over time as new CIM properties
 2238 are defined, the state sets are maintained in a separate document to allow them to be extended without
 2239 having to revise other PLDM specifications.

2240 25 Platform Descriptor Records (PDRs)

2241 PLDM can return collections of semantic and association information about the platform by using
 2242 collections of information called Platform Descriptor Records (PDRs). This information can include
 2243 records that return semantic information about sensors, such as their sensor resolution, tolerance,
 2244 accuracy, and conversion factors, as well as records that return information about the associations
 2245 between sensors and monitored entities, management controllers, effecters, and other platform
 2246 associations or capabilities.

2247 PDRs are called descriptor records because they are mainly used to describe the subsystem, rather than
 2248 to control it or configure it.

2249 25.1 PDR Repository updates

2250 A PDR Repository is not necessarily a static set of records. A platform that includes hot-plug devices or
 2251 supports field updates may have its PDRs change over time as devices are added or removed. Even if
 2252 the implementation of a particular platform management subsystem is static, the PDRs must still be
 2253 generated and installed so that they represent the semantic information and relationships of the particular
 2254 platform implementation.

2255 PLDM does not specify the mechanisms by which PDRs get generated, installed, or updated. This was
 2256 done intentionally to allow the vendor of the PDR Repository devices to create update or configuration
 2257 utilities that are appropriate for the particular implementation. PLDM does, however, specify how the
 2258 information is accessed and used.

2259 **25.2 Internal storage and organization of PDRs**

2260 The PLDM specifications do not place any requirements on how PDRs are internally stored or organized
 2261 within the device or devices that implement the PDR Repository. PDRs may be compressed, stored with
 2262 additional pointers, sorted, cross indexed, split, replicated, and so on, as long as the information meets
 2263 the byte order and formats specified for the PDR commands. The byte order and formats for PDRs are
 2264 specified in tables for the different PDR types in clause 28.

2265 **25.3 PDR types**

2266 PDRs are identified by a PDR Type value that is given in a field in the header for each different PDR.
 2267 PDR types include type values for records that identify PDRs for PLDM numeric and state sensors,
 2268 records that direct sensor initialization, records that describe PLDM effecters, and so on. The PDR Type
 2269 values are given in Table 76.

2270 **25.4 PDR record handles**

2271 All PDRs are assigned an opaque numeric value called the recordHandle. This value is used for
 2272 accessing individual PDRs within the PDR Repository. Additional information about recordHandles and
 2273 their use is provided in the specification of the GetPDR command (see 26.2).

2274 **25.5 Accessing PDRs**

2275 For most implementations, PDR data rarely changes. A party that uses PDR information may want to
 2276 cache certain information to reduce the need for accessing the PDR Repository. The
 2277 GetPDRRepositoryInfo command provides timestamps that can be used to identify whether any record
 2278 data in a particular PDR Repository has changed. If a change is detected the party can then update its
 2279 cached information as necessary.

2280 **26 PDR Repository commands**

2281 This clause describes the commands for accessing PDRs from a PDR Repository per this specification.
 2282 The command numbers for the PLDM messages are given in clause 30.

2283 If a PDR Repository is implemented, the Mandatory/Optional/Conditional (M/O/C) requirements shown in
 2284 Table 66 apply.

2285 **Table 66 – PDR Repository commands**

Command	M/O/C	Reference
GetPDRRepositoryInfo	M	See 26.1.
GetPDR	M	See 26.2.
FindPDR	O ^[1]	See 26.3.
RunInitAgent	C ^[2]	See 26.4.
GetPDRRepositorySignature	C ^[1]	See 26.5

2286 ^[1] Because this command reduces or eliminates the need to 'walk' the PDRs in order to find particular records, it is
 2287 recommended for Primary PDR Repositories that include multiple entity-association hierarchies, use a wide
 2288 range of PDR types, incorporate a large number of PDRs, or where specific PDRs, such as OEM PDRs, need
 2289 to be accessed by entities that do not care about other PDRs types.

2290 ^[2] The RunInitAgent command is required for the terminus that provides the primary PDR Repository.

2291 26.1 GetPDRRepositoryInfo command

2292 The GetPDRRepositoryInfo command returns information about the size and number of records in the
2293 PDR Repository of a particular PLDM terminus, and timestamps that indicate the last time that an update
2294 to the repository occurred. Two timestamps are returned: one that indicates whether any PLDM standard
2295 PDRs have changed, and another that indicates whether any OEM PDRs (if any) have changed.

2296 See 25.5 for more information about accessing PDRs. Table 67 describes the format of this command.

2297

Table 67 – GetPDRRepositoryInfo command format

Type	Request data
–	none
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES }</p>
enum8	<p>repositoryState</p> <p>value: { available, // Record data can be read from the repository. updateInProgress, // Record data is unavailable because an update is in progress. failed // Record data is unavailable because of a detected failure // condition. }</p>
timestamp104	<p>updateTime</p> <p>This timestamp identifies when the standard PDR Repository data was originally created, or the time of the most recent update if the data has been updated after it was created. This time does not include changes of PDRs that have a PDR Type of "OEM".</p>
timestamp104	<p>OEMUpdateTime</p> <p>This timestamp identifies when OEM PDRs in the PDR Repository were originally created, or the time of the most recent update if the data has been updated after it was created.</p>
uint32	<p>recordCount</p> <p>Total number of PDRs in this repository</p>
uint32	<p>repositorySize</p> <p>Size of the PDR Repository in bytes. This value provides information that can be used for helping estimate buffer size requirements when accessing PDRs.</p> <p>This size covers only the cumulative sizes of the PDR record fields. This size does not include the size for any internal header structures that are used for maintaining the PDRs. This number does not report and may not directly correlate to the amount of internal storage used for PDRs because, for example, an implementation may elect to internally compress or use other encodings of the PDR data.</p> <p>An implementation is allowed to round this number up to the nearest kilobyte (1024 bytes).</p>
uint32	<p>largestRecordSize</p> <p>Size of the largest record in the PDR Repository in bytes. This value provides information that can be used for helping estimate buffer size requirements when accessing PDRs.</p> <p>An implementation is allowed to round this number of up to the nearest 64-byte increment.</p>
uint8	<p>dataTransferHandleTimeout</p> <p>The minimum interval, in seconds, that a dataTransferHandle value remains valid after it was delivered in the response of a GetPDR or FindPDR command.</p> <p>special values: { 0x00 = no timeout, 0x01 = default minimum timeout (MC1, see clause 28.25), 0xFF = timeout >254 seconds. Any timeout values that are less than the specified default minimum timeout are illegal. }</p>

2298 **26.2 GetPDR command**

2299 The GetPDR command is used to retrieve individual PDRs from a PDR Repository. The record is
 2300 identified by the PDR recordHandle value that is passed in the request. The command can also be used
 2301 to dump all the PDRs within a PDR Repository.

2302 **26.2.1 GetPDR command format**

2303 Table 68 describes the format of the GetPDR command.

2304 **Table 68 – GetPDR command format**

Type	Request data
uint32	recordHandle The recordHandle value for the PDR to be retrieved. For more information, see 26.2.3 and 26.2.4. special value: {0x0000_0000 = Get first PDR in the repository}
uint32	dataTransferHandle A handle that is used to identify a particular multipart PDR data transfer operation. For more information, see 26.2.7 and 26.2.8. special value: { use 0x0000_0000 if the transferOperationFlag is GetFirstPart }
enum8	transferOperationFlag Indicates whether this request is for the first portion of the PDR value: { GetNextPart = 0x00, GetFirstPart = 0x01}
uint16	requestCount The maximum number of record bytes requested to be returned in the response to this instance of the GetPDR command. NOTE The responder may return fewer bytes than were requested.
uint16	recordChangeNumber value: If the transferOperationFlag field is set to GetFirstPart, set this value to 0x0000. If the transferOperationFlag field is set to GetNextPart, set this to the recordChangeNumber value that was returned in the header data from the first part of the PDR (see 28.1).
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_DATA_TRANSFER_HANDLE = 0x80, INVALID_TRANSFER_OPERATION_FLAG=0x81, INVALID_RECORD_HANDLE = 0x82, INVALID_RECORD_CHANGE_NUMBER = 0x83, TRANSFER_TIMEOUT = 0x84, REPOSITORY_UPDATE_IN_PROGRESS = 0x85 }
uint32	nextRecordHandle The recordHandle for the PDR that is next in the PDR Repository. The value can be used as the recordHandle in a subsequent GetPDR command as a means of sequentially reading PDRs from the repository. PDRs are not required to be returned in any particular order. special value: { 0x0000_0000 = no more PDRs following this one. }
uint32	nextDataTransferHandle A handle that identifies the next portion of the PDR data to be transferred, if any portions are remaining special value: { returns 0x0000_0000 if there is no remaining data. }

Type	Response data (continued)
enum8	<p>transferFlag</p> <p>Indicates what portion of the PDR is being transferred</p> <p>value: {Start = 0x00, Middle = 0x01, End = 0x04, StartAndEnd = 0x05}</p>
uint16	<p>responseCount</p> <p>The number of recordData bytes returned in this response</p> <p>special value: { returns 0x0000 if the requestCount was 0x0000 }</p>
(var)	<p>recordData</p> <p>PDR data bytes. This field is absent if responseCount = 0x0000. The number of PDR data bytes returned in this field must match responseCount.</p>
<i>If transferFlag = End</i>	
uint8	<p>transferCRC</p> <p>A CRC-8 for the overall PDR. This is provided to help verify data integrity for a PDR when it is transferred using a multipart transfer. The CRC is calculated over the entire PDR data using the polynomial $x^8 + x^2 + x^1 + 1$ (This is the same polynomial used in the MCTP over SMBus/I²C transport binding specification). The CRC is calculated from most-significant bit to least-significant bit on bytes in the order that they are received. This field is only present when transferFlag = End.</p>

2305 **26.2.2 Single-part and multipart transfers**

2306 The data from a given PDR may be accessed using a single-part or multipart transfer. A single transfer
 2307 occurs when the entire PDR content is delivered using a single GetPDR command response. A multipart
 2308 transfer is required either when the record data exceeds the amount of data that the responder can return
 2309 using a single response, or when it exceeds the amount of data that the requester can accept in a single
 2310 response. In this case, the GetPDR command is used iteratively to retrieve the first portion of the record
 2311 and then subsequent portions. Additional information and requirements for multipart transfers is provided
 2312 in 26.2.7.

2313 Partial transfers from the beginning of a record are allowed. That is, a requester is not required to read
 2314 out an entire record if only the beginning portion of the record data is of interest.

2315 **26.2.3 PDR recordHandle**

2316 The recordHandle is an opaque value that is used by the implementation of the PDR Repository to
 2317 identify individual records and to track where the next data of a multipart transfer will come from. This
 2318 value is obtained from the response data of a previous instance of the GetPDR command. A special
 2319 value of 0x0000_0000 is used to retrieve the first PDR in the repository.

2320 Some implementations may use the recordHandle as a direct offset into storage memory, others may use
 2321 it as offset that is relative to the start of the PDR data, and others may use it as a table or list index.

2322 **26.2.4 PDR recordHandle retention**

2323 The recordHandle values that are used to access a particular PDR may change when the
 2324 recordChangeNumber is changed. recordHandle values are also not guaranteed to endure across
 2325 connections to the given PLDM terminus that is implementing the command. A party that needs to re-
 2326 establish a connection to the terminus must assume that any PDR recordHandle values that it previously
 2327 had are no longer valid. If any multipart transfers were not completed before the connection was re-
 2328 established, those transfers must be restarted from the beginning.

2329 **26.2.5 PDR recordChangeNumber**

2330 The recordChangeNumber provides a mechanism for preventing the use of invalid PDR data if a record's
2331 data gets updated while the record was in the process of being read out. The mechanism helps ensure
2332 that a requester does not get the first parts from an earlier version of the record and remaining parts from
2333 a later version of the record. The recordChangeNumber can also be used to help a requester scan and
2334 identify which PDRs may have changed after an update to the PDR Repository has occurred.

2335 To accomplish this, the PDR recordChangeNumber that is returned in the GetPDR response is required
2336 to change whenever the data of a PDR changes during a multipart access of the PDR. The party that is
2337 accessing a PDR gets the recordChangeNumber when the first part of the record is returned. This
2338 number is then used as one of the input parameters when retrieving the remaining parts of the record.

2339 The PLDM responder compares this number against the present recordChangeNumber that is associated
2340 with the record. If there is a mismatch, the PLDM responder returns an error completionCode. The
2341 requester can then handle the error by starting the PDR transfer over.

2342 It is recommended that an implementation update the recordChangeNumber only for records that have
2343 changed due to an update. However, implementations may elect to update the recordChangeNumber for
2344 some or all unchanged records. This latter approach can be used for small and simple implementations in
2345 which PDR exits and updates are rare, but should be avoided in large implementations in which the party
2346 that is accessing the PDR data may see significant delays due to the unnecessary re-reading and
2347 handling of PDRs that have not actually changed.

2348 **26.2.6 PDR Repository timestamp and PDR Repository locking**

2349 The recordChangeNumber mechanism protects against inconsistent data only on a per record basis; it
2350 does not automatically protect against inconsistencies that may occur due to individual updates of
2351 interrelated records. For example, if record A and B are interrelated and both need synchronized updates,
2352 it is possible that a party could access the records at a time when A has been updated but B has not. The
2353 individual records would be correct, but their interrelationship could be incorrect.

2354 The party that is updating the PDRs can lock the repository while updates are occurring (the mechanisms
2355 used for updating and locking the PDRs are outside this specification). In this case, commands such as
2356 the GetPDR command will return an error completionCode indicating that the repository records are
2357 inaccessible because an update is in progress. Update-in-progress status is also available in the
2358 GetPDRRepositoryInfo command.

2359 A party that updates records in a PDR Repository while PLDM command handling is active must either:
2360 lock the PDRs and update the timestamp and recordChangeNumber values before making the repository
2361 available; or update the timestamp and recordChangeNumber values as each individual updated record
2362 is made available through PLDM.

2363 The PDR Repository has a timestamp that can be read using the GetPDRRepositoryInfo command. The
2364 timestamp value is updated whenever changes are made to the repository. A party that is accessing
2365 multiple PDRs and relying on an interrelationship between those records should check the timestamp
2366 value after retrieving the records to verify that a repository update did not occur while the records were
2367 being accessed.

2368 If an update has occurred while records were being read, the records should either be re-read or have
2369 their recordChangeNumber values checked to see if they have changed. Because the
2370 recordChangeNumber is in the beginning portion of a PDR, it is not necessary to read the entire record to
2371 get the value.

2372 **26.2.7 Multipart PDR transfers**

2373 The command is intended to support multipart transfer of PDR data only in a sequential manner, starting
2374 from the beginning of the PDR. Random access to a middle portion of a PDR is not required by
2375 implementations, nor is it intentionally supported as an option in this specification.

2376 The dataTransferHandle value is therefore required to remain valid only for use with the next GetNextPart
2377 operation from a given requester. Although many implementations will likely return the same data for an
2378 identical sequence of PDR access commands regardless of the ID of the requester, an implementation
2379 may allocate and track dataTransferHandles on a per-requester basis. The dataTransferHandle
2380 information given to one requester might not be usable by another requester.

2381 **26.2.8 PDR dataTransferHandle retention**

2382 The dataTransferHandle value for a multipart transfer is required to remain valid for at least MC1 seconds
2383 after it has been delivered in a response. After this interval, an implementation may elect to implement a
2384 timeout and terminate the multipart transfer. To support this, an implementation would use some aspect
2385 of the recordHandle value to track the particular multipart transfer in progress.

2386 The provisions that allow a dataTransferHandle value to become invalid or expire allow implementations
2387 the option of temporarily queuing PDR data in memory and freeing up that memory if the record data is
2388 no longer being accessed. The provisions eliminate the need for the recordHandle values for a given
2389 request to remain valid indefinitely.

2390 **26.2.9 Multipart PDR transfer termination and timeouts**

2391 No formal release mechanism exists for multipart PDR transfers. Multipart transfers may be terminated by
2392 the responder under the following conditions:

- 2393 • The responder implementation may restrict a given requester to having only one PDR transfer
2394 in process at a time. If the requester starts a different transfer, the earlier multipart transfer that
2395 was in progress may be aborted.
- 2396 • The responder implementation may terminate any multipart PDR transfer in progress following
2397 expiration of the PDR dataTransferHandle retention interval, MC1.
- 2398 • Execution of the Initialization Agent function may terminate a multipart PDR transfer in progress.

2399 **26.2.10 Reuse of prior request values**

2400 Except for the first part of a PDR, an implementation is not required to support returning a previously
2401 transferred portion of a PDR after the transfer has progressed to a later portion. For example, if the first
2402 three portions of a PDR have been transferred, the implementation may not allow a re-transfer of the
2403 second portion without restarting the transfer from the beginning. If an implementation does accept
2404 request parameters that were used for reading an earlier portion of a given PDR, it must return the same
2405 PDR data that was returned for the original request.

2406 **26.3 FindPDR command**

2407 The FindPDR command is provided to improve the efficiency of common types of access to a Primary
2408 PDR Repository. The FindPDR command is primarily designed to provide operations that can assist a
2409 MAP in using information from the PDRs to instantiate CIM objects and associations.

2410 The FindPDR command returns the PLDMHandleType and PLDMHandle values for a particular PDR or
2411 set of PDRs, depending on the parameters that were passed in the request. The response can also
2412 include the first portion of the PDR data. The response from the FindPDR command can then be used
2413 with the GetPDR command to read the PDR or the remaining portions of the PDR.

2414 To reduce implementation and validation complexity, the FindPDR command does not provide a generic
 2415 search engine but supports only a limited number of different preconfigured queries that are restricted to
 2416 using particular key fields within the PDRs.

2417 For example, the FindPDR command can be used to find all the PDRs that have a particular
 2418 PLDMTerminusHandle, or Entity Association PDRs that have a common Container ID. It can also be used
 2419 to find Numeric Sensor PDRs that share a particular type of monitored numeric unit, such as temperature,
 2420 or state sensors that use a particular state set. However, the FindPDR command does not support less
 2421 common operations such as finding records that have a particular hysteresis value setting or state
 2422 sensors that implement a particular state from within a state set.

2423 The findParameters field holds the PDRTYPE-specific search fields. The format of findParameters is
 2424 identified by the parameterFormatNumber that is passed in the request. The findParameters value may
 2425 be applicable to more than one PDRTYPE. The parameterFormatNumber and PDRTYPE field in the
 2426 request are used together to identify which PDRs should be searched. Table 70 lists the values for
 2427 parameterFormatNumber and the PDRTYPE values that are associated with each
 2428 parameterFormatNumber. Table 71 lists the different PDR fields that make up the findParameters value
 2429 for each different parameterFormatNumber.

2430 If the PDRTYPE field value is set to 0, all of the PDRTYPE values that are specified for the
 2431 parameterFormatNumber in Table 70 are searched. Otherwise, only PDRs that have the given PDRTYPE
 2432 value are searched.

2433 For example, if PDRTYPE = 0 and parameterFormatNumber = 7, all PDRs with PDRTYPE values that are
 2434 identified for searching with parameterFormatNumber = 7 are searched: Numeric Effector Initialization,
 2435 State Effector Initialization, and Effector Auxiliary Names. If the PDRTYPE is set to the value for State
 2436 Effector Initialization PDR, only State Effector Initialization PDRs are searched.

2437 The findParameters value is included in each request to eliminate the need for implementations to retain
 2438 the findParameters value when a multi-PDR find operation is being done.

2439 Table 69 describes the format of this command.

2440 **Table 69 – FindPDR command format**

Type	Request data
uint32	<p>findHandle</p> <p>A handle that is used to track the point from which searching should resume. With the exception of the first find, the nextFindHandle value is set with the nextFindHandle value from the previous response for the find operation in process.</p> <p>special values: { use 0x0000_0000 if the findOperation is findFirst, 0xFFFF_FFFF = reserved. }</p> <p>NOTE: This field has the same retention specifications as the dataTransferHandle field used in the GetPDR command. See 26.2.4 for more information.</p>
enum8	<p>findOperationFlag</p> <p>Indicates whether this request is for locating the first matching PDR.</p> <p>value: { findNext = 0x00, findFirst = 0x01 }</p>
uint16	<p>requestCount</p> <p>The maximum number of record bytes requested to be returned in the response to this instance of the FindPDR command.</p> <p>NOTE: The responder may return fewer bytes than were requested.</p>

Type	Request data (continued)
uint16	<p>PDRType</p> <p>The PDRType for the records to be located.</p> <p>special value: 0x0000 = match any PDRType.</p>
uint8	<p>parameterFormatNumber</p> <p>A number that identifies the format and number of parameters in the findParameters field. Table 71 lists the different PDR fields that make up the findParameters value for each different parameterFormatNumber.</p>
bitfield16	<p>wildcards</p> <p>Each Nth bit position indicates whether the Nth parameter from the findParameters field should be matched or ignored (treated as a wildcard). Use 0b for any bit position for which a parameter is not defined.</p> <p>[15] – 1b = sixteenth parameter value in findParameters must be matched 0b = sixteenth parameter value in findParameters is ignored</p> <p>...</p> <p>[0] – 1b = first parameter value in findParameters must be matched 0b = first parameter value in findParameters is ignored</p>
varies	<p>findParameters</p> <p>A series of parameters that correspond to fields in the PDRs that are used for the find operation.</p> <p>Table 71 lists the PDR fields that make up the findParameters value for each parameterFormatNumber. Each field within findParameters is provided in the order listed in Table 71, starting from the top of the table to the bottom for the column that is identified by parameterFormatNumber. Dots in the column identify which parameters are to be provided in findParameters. The data type and size (for example, uint8) and meaning of each parameter are given by the definition of the PDR that is identified by the PDRTypes for the given parameterFormatNumber, as listed in Table 70.</p> <p>Values for all parameters must be provided even if a particular parameter is to be ignored in the search. The values for ignored parameters shall not be checked for validity by the responder. An implementation may optionally check non-wildcard parameters for validity and return an error completionCode if the parameter is not a legal value for the corresponding field in the PDR.</p>

Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, INVALID_FIND_HANDLE = 0x80, INVALID_FIND_OPERATION_FLAG = 0x81, INVALID_PDR_TYPE = 0x82, INVALID_PARAMETER_FORMAT_NUMBER = 0x83, INVALID_FIND_PARAMETERS = 0x84, REPOSITORY_UPDATE_IN_PROGRESS = 0x85 }</p>
uint32	<p>nextFindHandle</p> <p>A handle that identifies the next part of a Find operation that may return more than one PDR. The implementation uses this field to track the point from which it needs to resume searching. An implementation may elect to look ahead to see if there are any more matching PDRs before sending the response, or it may elect to wait until getting the next request before searching to see if there are any remaining matching records. The “look-ahead” approach is recommended.</p> <p>special values: { returns 0x0000_0000 if no matching PDR was found. returns 0xFFFF_FFFF if this response holds data for the last matching PDR. That is, there are no more matching PDRs beyond this one.}</p>
uint32	<p>nextDataTransferHandle</p> <p>A handle that identifies the next portion of the PDR data to be transferred, if any portions are remaining. This value is used in the GetPDR command to retrieve any remaining portions of the PDR.</p> <p>special value: { returns 0x0000_0000 if there is no remaining recordData beyond the recordData that is being returned in this response data. }</p>
enum8	<p>transferFlag</p> <p>Indicates what portion of the PDR is being transferred</p> <p>value: {Start = 0x00, Middle = 0x01, End = 0x04, StartAndEnd = 0x05}</p>
uint16	<p>responseCount</p> <p>The number of recordData bytes returned in this response</p> <p>special value: { returns 0x0000 if the requestCount was 0x0000 }</p>
(var)	<p>recordData</p> <p>PDR data bytes. This field is absent if responseCount = 0x0000. Otherwise, the number of PDR data bytes returned in this field must match responseCount.</p>

2441

Table 70 – FindPDR Command Parameter Format Numbers

PDRType	parameterFormatNumber
ANY = 0	1 ^[1]
Event Log	1 ^[2]
Terminus Locator	2
Numeric Sensor	3
Numeric Sensor Initialization	4
State Sensor Initialization	
Sensor Auxiliary Names	
State Sensor	5
Numeric Effector	6
Numeric Effector Initialization	7
State Effector Initialization	
Effector Auxiliary Names	
State Effector	8
Entity Association	9
Interrupt Association	10
OEM Unit	11
OEM State Set	12
OEM Entity	13
OEM Device	14
OEM	
OEM Unit	15 ^[3]
OEM State Set	
OEM Entity	
OEM Device	
OEM	

2442 ^[1] The entire contents of the repository can be read by using this format along with PDRType = ANY and PLDMTerminusHandle set
 2443 for "wildcard."

2444 ^[2] The PLDMTerminusHandle parameter must be set for "wildcard" when using this format to search for Event Log PDRs.

2445 ^[3] This search format can be used to return all PDRs that have any of the indicated "OEM" PDRType values or all PDRs that have
 2446 any of the indicated "OEM" PDRType values and match a particular vendorIANA.

Table 71 – FindPDR command parameter formats

Parameter (PDR field)	parameterFormatNumber														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PLDMTerminusHandle	•	•	•	•	•	•	•	•		•	•	•	•	•	•
TID		•													
sensorID			•	•	•					•					
effectorID						•	•	•							
stateSetID					•			•							
containerID			•			•		•	•						
associationType									•						
entityType			•			•									
entityInstanceNumber			•			•									
baseUnit			•			•									
unitModifier			•			•									
rateUnit			•			•									
baseOEMUnitHandle			•			•									
auxUnit			•			•									
auxUnitModifier			•			•									
auxrateUnit			•			•									
auxOEMUnitHandle			•			•									
containerEntityType									•						
containerEntityInstanceNumber									•						
containerEntityEntityID									•						
interruptTargetEntityType										•					
interruptTargetEntityInstanceNumber										•					
interruptTargetEntityContainerID										•					
interruptSourceEntityType										•					
interruptSourceEntityInstanceNumber										•					
interruptSourceEntityContainerID										•					
OEMUnitHandle											•				
OEMStateSetIDHandle												•			
OEMEntityIDHandle													•		
vendorIANA											•	•	•	•	•
OEMUnitID											•				
OEMStateSetID												•			
OEMEntityID													•		
OEMRecordID														•	

2448 **26.4 RunInitAgent command**

2449 The RunInitAgent command directs the terminus that provides the Primary PDR Repository to run the
 2450 Initialization Agent function. This command can be used to trigger a reinitialization of the monitoring and
 2451 control capabilities in the PLDM subsystem. Table 72 describes the format of the command.

2452 **Table 72 – RunInitAgent command format**

Type	Request data
bitfield8	<p>initConditionEmulation</p> <p>This value selects a condition that emulates a transition that triggers the Initialization Agent to run. The Initialization Agent then performs its steps accordingly. For example, if the initConditionEmulation is set to SystemHardReset, the Initialization Agent initializes only those sensors and effecters that have SystemHardReset set in the initCondition parameter of their Initialization PDRs.</p> <p>value: {</p> <p> 0x00 = InitializationAgentRestart, // Directs the Initialization Agent to take the same steps // as it would if the controller that holds the Initialization // Agent was restarted or reinitialized.</p> <p> 0x01 = PLDMSubsystemPowerUp, // Directs the Initialization Agent to take the same steps // as it would when the PLDM subsystem becomes // powered up.</p> <p> 0x02 = SystemHardReset, // Directs the Initialization Agent to take the same steps // as it would following a system hard reset.</p> <p> 0x03 = SystemWarmReset, // Directs the Initialization Agent to take the same steps // as it would following a system warm reset.</p> <p> 0x04 = PLDMTerminusOnline // Directs the Initialization Agent to initialize the // terminus that has a TID that matches the TID // parameter in this request.</p> <p>}</p>
uint8	<p>TID</p> <p>Terminus ID for the terminus to be initialized when the initConditionEmulation field in this request is set to PLDMTerminusOnline.</p> <p>special value: The value in this field is ignored when the initConditionEmulation field in this request is set to any value other than PLDMTerminusOnline.</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES }</p>

2453 **26.5 GetPDRRepositorySignature command**

2454 The PDR Repository Signature is a value that represents the entire collection of terminus Platform Device
 2455 Records (PDRs). This is different than the GetPDRRepositoryInfo command because only an opaque 32
 2456 bit value is returned. The purpose of the PDR Repository Signature is to provide the management
 2457 controller the capability to determine whether a terminus PDR repository has changed during state
 2458 transitions such as power cycles. The PDR Repository signature shall remain persistent unless there is a
 2459 change in any PDR. This allows the management controller to not retrieve large number of PDRs if the
 2460 management controller caches the specific terminus PDR repository. The terminus is allowed to create
 2461 the PDR Repository Signature using any method that creates unique values to indicate a change. The

2462 management controller is expected to compare the current value to the previous value to detect a
 2463 terminus PDR Repository modification.

2464 **Table 73 – GetPDRRepositorySignature command format**

Type	Request data
--	none
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
uint32	pdrRepositorySignature This is a 32 bit value and remains persistent unless a change is detected in any record of the PDR repository. The supplier of the PDR Repository may choose the best method to create at least two different values. The receiver of the PDR Repository simply checks for a difference between previous pdrRepositorySignature and current pdrRepositorySignature to detect a change or update to the repository.

2465 **27 PDR definitions**

2466 This clause describes certain important characteristic parameters that are provided within the PDRs for
 2467 interpreting the readings and settings of sensors and effecters.

2468 **27.1 Sensor types**

2469 PLDM contains two basic types of sensors that are described using PDRs:

- 2470 • The PLDM Numeric Sensor is used to obtain a numeric value for a monitored parameter. The
 2471 sensor definition also optionally includes returning state information based on whether the
 2472 numeric reading has crossed one or more defined threshold levels.
- 2473 • The PLDM State Sensor/PLDM Composite State Sensor is used to obtain the present state of a
 2474 monitored parameter. The PLDM sensor access commands allow an implementation to provide
 2475 multiple sets of state information using a single access command. When this is done, the
 2476 implementation is referred to as providing a Composite State Sensor.

2477 **27.2 Effector types**

2478 PLDM contains two basic types of effecters that are described using PDRs:

- 2479 • The PLDM Numeric Effector is used to set a numeric value for a monitored parameter.
- 2480 • The PLDM State Effector/PLDM Composite State Effector is used to set the present state of a
 2481 monitored parameter. The PLDM effector access commands allow an implementation to provide
 2482 multiple sets of state information using a single access command. When this is done, the
 2483 implementation is referred to as providing a Composite State Effector.

2484 **27.3 State sets**

2485 State information is returned using an enumeration called a “state set.” Each state set has a different ID
 2486 number. This number is used within the PDRs to identify what particular state set a sensor or effector is
 2487 using. See clause 24 for more information.

2488 27.4 Sensor and effector units

2489 This subclause and following subclauses describe the fields that are used within PDRs to define and
2490 describe sensor and effector units and related characteristics such as accuracy, tolerance, and resolution.

2491 The type of units that are associated with the value that a sensor returns or monitors, or that an effector
2492 controls, such as volts or amps, is identified in the PDRs by a sensorUnits enumeration, listed in Table
2493 74. Unless otherwise indicated, the units apply to all numeric properties of the sensor, such as the sensor
2494 reading, threshold values, and resolution.

2495 Vendor-defined units are identified by a special value for OEMUnit. A special PDR called the OEM Unit
2496 PDR is used to define the meaning of the OEMUnit when it is used in the PDRs that describe a sensor or
2497 effector. Refer to 28.9 for more information about how OEMUnits are used in PDRs.

2498

Table 74 – sensorUnits enumeration

0	None	30	Cubic Feet	60	Bits
1	Unspecified	31	Meters	61	Bytes
2	Degrees C	32	Cubic Centimeters	62	Words (data)
3	Degrees F	33	Cubic Meters	63	DoubleWords
4	Kelvins	34	Liters	64	QuadWords
5	Volts	35	Fluid Ounces	65	Percentage
6	Amps	36	Radians	66	Pascals
7	Watts	37	Steradians	67	Counts
8	Joules	38	Revolutions	68	Grams
9	Coulombs	39	Cycles	69	Newton-meters
10	VA	40	Gravities	70	Hits
11	Nits	41	Ounces	71	Misses
12	Lumens	42	Pounds	72	Retries
13	Lux	43	Foot-Pounds	73	Overruns/Overflows
14	Candelas	44	Ounce-Inches	74	Underruns
15	kPa	45	Gauss	75	Collisions
16	PSI	46	Gilberts	76	Packets
17	Newtons	47	Henries	77	Messages
18	CFM	48	Farads	78	Characters
19	RPM	49	Ohms	79	Errors
20	Hertz	50	Siemens	80	Corrected Errors
21	Seconds	51	Moles	81	Uncorrectable Errors
22	Minutes	52	Becquerels	82	Square Mils
23	Hours	53	PPM (parts/million)	83	Square Inches
24	Days	54	Decibels	84	Square Feet
25	Weeks	55	DbA	85	Square Centimeters
26	Mils	56	DbC	86	Square Meters
27	Inches	57	Grays	-	all other = reserved
28	Feet	58	Sieverts		
29	Cubic Inches	59	Color Temperature Degrees K	255	OEMUnit

2499 27.4.1 Base units

2500 The base unit of measurement that is associated with the reading values returned by a PLDM Numeric
 2501 Sensor or set into a PLDM Numeric Effector is represented by the combination of three fields from the
 2502 PDR for the sensor: baseUnits, unitModifier, and rateUnits. These fields are interpreted according to the
 2503 following formula:

$$2504 \quad \text{Sensor/Effector Units} = \text{baseUnit} * 10^{\text{unitModifier}} \text{rateUnit}$$

2505 For example, if baseUnits is Volts and the unitModifier is -6, the units of the values returned are
 2506 microvolts.

2507 If the rateUnits property is set to a value other than None, the units are further qualified as rate units. In
 2508 the preceding example, if rateUnits is set to Per Second, the values returned by the sensor are in
 2509 microvolts/second.

2510 27.4.2 Auxiliary units

2511 In some cases, additional modification of the base unit of the sensor might be required. For example,
 2512 acceleration is commonly given in units such as "meters per second per second". The PDRs include a
 2513 provision for modifying the base units with an additional set of units called auxiliary units. Auxiliary units
 2514 are defined by three elements: auxUnit, auxUnitModifier, and auxRateUnit. These elements are used in
 2515 combination with the base units as follows:

$$2516 \quad \text{Sensor/Effector Units} = \text{baseUnit} * 10^{\text{unitModifier}} [\text{rel}] \text{auxUnit} * 10^{\text{auxUnitModifier}} \text{rateUnit auxRateUnit}$$

2517 [rel] is the relationship between the base unit and the auxiliary unit, as follows:

2518 rel = enum8 { dividedBy, multipliedBy }

2519 And:

2520 dividedBy implies a "/" or "per" relationship, such as "per foot"

2521 multipliedBy implies a "*" operation, such as "foot*lbs (foot-lbs)"

2522 auxUnit and auxRateUnit shall not be used if an equivalent definition can be made using only base units.

2523 27.4.3 Units for use with CIM

2524 Developers are cautioned that PLDM units may include types of units that are not presently supported by
 2525 standard CIM objects such as CIM_Sensor. PLDM supports additional types of units because certain
 2526 types of sensors or effectors may be used within a platform management subsystem but are not exposed
 2527 through CIM, or are mapped into CIM using proprietary CIM extensions. Parties developing platform
 2528 management subsystems in which sensors are intended to be exposed as CIM objects should first verify
 2529 which types of sensors and units are supported by CIM and the CIM profiles.

2530 27.4.4 OEM (vendor-defined) sensor units

2531 OEM (vendor-defined) sensor units are identified in PLDM sensor PDRs when the OEMUnit value from
 2532 Table 74 is used for the baseUnit or auxUnit. The semantic information of an OEMUnit can then be
 2533 further described using an OEM Sensor Units PDR that is associated with the particular sensor that is
 2534 returning the OEMUnit. Multiple OEM Sensor Units PDRs can be defined if there is a need for defining
 2535 more than one type of OEM unit. Additionally, multiple PLDM Sensor PDRs can be associated with a
 2536 particular OEM Sensor Units PDR.

2537 27.5 Counters

2538 A counter is a numeric sensor that returns a value that returns a count. PLDM does not define any
2539 requirements on whether a counter must increment, decrement, or both, or whether it does so
2540 sequentially or monotonically, and so on.

2541 Many common types of counters can use predefined sensor unit values, such as Hits, Misses, Corrected
2542 Errors, Uncorrected Errors, and others. If no predefined unit fits, it is recommended that the auxiliary
2543 sensor unit (auxUnit) be designated using the predefined unit "Counts" in the PDR for the sensor, and
2544 that an OEM unit type is defined for the base unit.

2545 For example, if an implementation needed a counter for "widgets," it would be noted that no predefined
2546 sensor unit type for "widgets" exists. In this case, an OEM Unit PDR for "widgets" is created and used for
2547 the base unit type, and "Counts" is used as the auxUnit.

2548 Counters enable a party that accesses PDR information for the sensor to get a partial interpretation of the
2549 sensor semantics. Thus, although the party interpreting the sensor may not know what a widget is, it will
2550 know that the sensor is returning Counts of something.

2551 27.6 Accuracy, tolerance, resolution, and offset

2552 The PDRs for numeric sensors and effecters include fields for reporting the accuracy, tolerance, and
2553 resolution associated with the numeric value for the reading or setting. This subclause provides
2554 definitions for accuracy, tolerance, and resolution as used within this specification and information on how
2555 the values are calculated and used. Accuracy, tolerance, and resolution are summarized as follows:

2556 **Accuracy** An error in the reading that scales proportionally with the magnitude of the input. Typically
2557 given as a \pm percentage of the reading.

2558 **Tolerance** A \pm error in the reading that, unlike accuracy, does not scale with the magnitude of the
2559 reading. Tolerance typically comes from a combination of quantization (round off) errors
2560 including errors due to offsets in the measurement.

2561 **Resolution** The nominal size of the "steps" between sequential reading values.

2562 Accuracy specifies a degree of error that varies in proportion to the reading, and tolerance specifies a
2563 constant error. The combination of these two generally provides enough flexibility to cover a range of
2564 conversion errors in most linear analog-to-digital (A/D) converters.

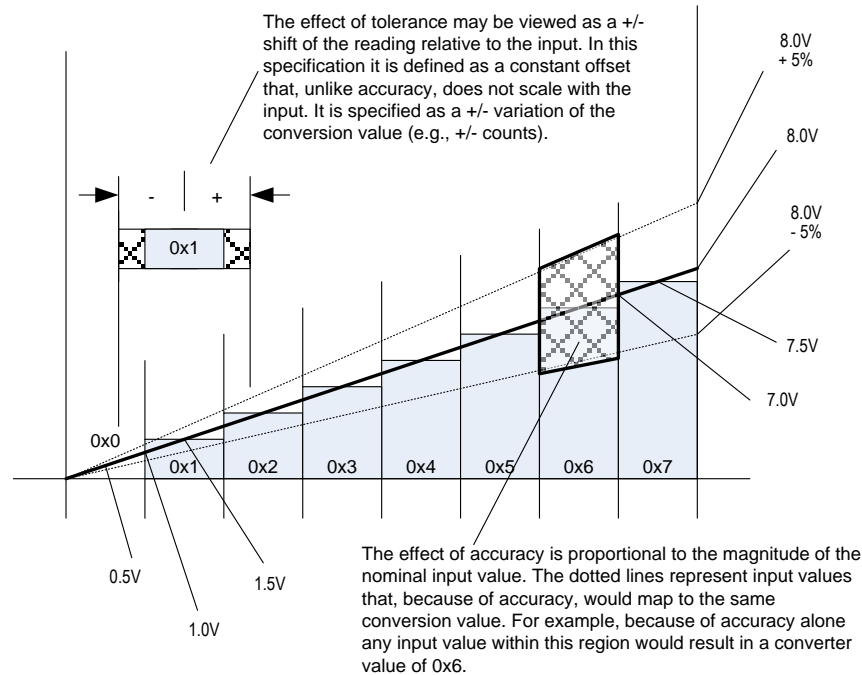
2565 Although other error types, such as nonlinearity, can exist in converters, the contribution of those errors
2566 can be accounted for by increasing the size of the reported values for tolerance, accuracy, or both as
2567 necessary.

2568 27.6.1 Additional information about numeric sensor/effector tolerance

2569 Tolerance can be considered to be a constant portion of the quantization error in the conversion of an
2570 analog input to a numeric sensor. Consider a sensor where 0x00 ideally corresponds to 0.000 to 0.500 V
2571 and 0x01 corresponds to 0.500 V to 1.000 V. When the input is 0.500 V exactly, the sensor could report
2572 either 0x00 or 0x01. Now assume that the input is 0.501 V. Ideally, this would result in a value of 0x01
2573 from the sensor, but because of offsets in an implementation, it is possible that some implementations
2574 could return a value of either 0x00 or 0x01. If 0x00 is reported, the sensor is effectively returning a value
2575 that is -1 count from ideal. It is possible that the sensor implementation could be asymmetric with respect
2576 to tolerance. For example, a sensor implementation may sometimes map 0.501 V to 0x00, but would
2577 never map anything less than 0.500 V to 0x01. In this case, the tolerance would be +0 counts and -1
2578 counts. Generally, an implementation is subject to both positive and negative offsets because of
2579 component manufacturing variation, noise, and so on. Thus, it is common to see a tolerance of ± 1 count.

2580 **27.6.2 Examples of accuracy, tolerance, and resolution use**

2581 Figure 24 shows an example of a "3-bit" (eight step) converter. In this example, the converter is hooked
 2582 up for monitoring a nominal signal that can vary from 0.0 V to 8.0 V. The resolution is defined as the size
 2583 of the steps between nominal readings. The resolution is 1.0 V because there is 1.0 V difference between
 2584 each successive reading value.



2585

2586 **Figure 24 – Accuracy, tolerance, and resolution example**

2587 In this example, the input value that corresponds to a reading of 0x0 is actually centered around 0.50 V,
 2588 not 0.0 V. That is, the meaning of a reading of 0x0 does not mean 0.0 V, as might be expected, but
 2589 actually means "0.5 V plus or minus 0.5 V". This represents a typical way that A/D converters are
 2590 connected in systems. It is a common mistake to assume that a reading of zero actually corresponds to
 2591 0.0 V.

2592 If this converter had no additional offsets or accuracy errors, the reading values would correspond to input
 2593 values as follows:

- 2594 0x0 → 0 V to 1.0 V (0.5 V ± 0.5 V)
- 2595 0x1 → 1.0 V to 2.0 V (1.5 V ± 0.5 V)
- 2596 0x2 → 2.0 V to 3.0 V (2.5 V ± 0.5 V)
- 2597 0x3 → 3.0 V to 4.0 V (3.5 V ± 0.5 V)
- 2598 0x4 → 4.0 V to 5.0 V (4.5 V ± 0.5 V)
- 2599 0x5 → 5.0 V to 6.0 V (5.5 V ± 0.5 V)

2600 0x6 → 6.0 V to 7.0 V (6.5 V ± 0.5 V)

2601 0x7 → 7.0 V to 8.0 V (7.5 V ± 0.5 V)

2602 If these readings were converted to their corresponding nominal input voltage (V_{in}) values, the formula
2603 would be as follows:

2604 $V_{in}(\text{nominal}) \rightarrow (\text{resolution} * \text{reading}) + 1/2 \text{ resolution}$

2605 Note that this follows the Cartesian coordinate formula for a line: $y = Mx + B$

2606 Now, suppose that the implementation could add a negative D.C. offset of 0.5 V to the input. Then the
2607 center point for a reading of 0.0 V would correspond to 0.0 V, and a reading of 0x0 would correspond to a
2608 range of 0.0 V ± 0.5 V instead of 0.0 V to 1.0 V. In this case, the conversion would then be $V = (\text{resolution}$
2609 $* \text{reading}) + 0.0 \text{ V}$. There is now no offset relative to the center of the reading value because of a D.C.
2610 offset. If the converted negative offset of 4.0 V was connected to the input, a reading of 0x0 would now
2611 correspond to -3.5 V ± 0.5 V and a reading of 111b would correspond to 3.5 V ± 0.5 V.

2612 It is very common for an A/D converter implementation to have a D.C. offset that needs to be accounted
2613 for when converting a reading to the corresponding nominal input value. The party that implements the
2614 hardware for the sensor needs to provide this offset value as well as the resolution (step size per count)
2615 so that the basic conversion of the reading can be accomplished.

2616 After the basic conversion of the reading is done, the effects of accuracy and tolerance may need to be
2617 taken into account. For example, if someone is depending on the reading to determine whether
2618 something has failed, it is important to understand how much error might be in the reading so that a
2619 failure is not falsely assessed for a healthy component.

2620 For PLDM, the effects of accuracy and tolerance are considered to be orthogonal to one another and
2621 additive. First consider the effect of accuracy. Suppose the accuracy of the sensor is specified as ±5%.
2622 Using that figure, a value of 001b will nominally correspond to 1.5 V ± 5%, but because of quantization
2623 and accuracy, any value from 1.0 V ± 5% to 2.0 V ± 5% (a range of 0.95 V to 2.10 V) could result in a
2624 reading of 0x1.

2625 The next step is to factor in tolerance. The quantization within a converter is never perfect; some slight
2626 variation always exists in the comparison points that yield a particular converter output. Instead of the
2627 conversion ranges being evenly spaced as shown in Figure 24, some ranges may be a little wider and
2628 others a little narrower. The effect of this is that in an actual implementation, borderline values such as
2629 1.99 V or 2.01 V, for example, may sometimes yield a value of 0x1 and sometimes 0x2.

2630 Tolerance in PLDM is defined as an error in the quantization that is applied to all counts of the converter
2631 equally. Because PLDM sensors are all specified as returning integer values, any errors in the reading
2632 will always result in an integral number of counts. Thus, tolerance is specified as a +/- effect on the count.

2633 The tolerance value is typically used to account for quantization errors in A/D conversion circuitry that
2634 occur because of effects such as D.C. voltage offsets within the circuit. For example, suppose the input to
2635 an A/D converter that monitors voltage was shifted up by a constant amount, as would be the case if a
2636 D.C. offset was added to the input. Per the figure, if a D.C. offset error of 0.25 V were added when
2637 converting, the input reading 0x01 would represent a range that actually goes from 0.75 V to 1.75 V
2638 instead of the nominal range 1.0 V to 2.0 V. This means that an input between 0.75 V and 1.0 V will
2639 cause a reading of 0x1 to be returned instead 0x0. Thus, because of this offset error, the reading would
2640 be one count higher than it was intended to be for inputs in that range. Similarly, with the same offset, a
2641 reading of 0x2 would correspond to an input of 1.75 V to 2.75 V, and so an input between 1.75 V and
2642 2.00 V would also result in a reading that is one count higher than intended.

2643 This does not mean that all conversions are off by one count. In this example, the reading is incorrect
2644 only for inputs that are in the range caused by the offset. A reading of 0x1 would be correctly returned for

2645 an input of 1.5 V. The reading can thus be incorrect by 0 counts or +1 counts depending on what range
2646 the input value is in. In this case, the tolerance would be specified as +1/-0 counts.

2647 Manufacturing variations and tolerances in A/D conversion circuitry mean that both positive and negative
2648 offsets are possible. This is why it is typical to see a specification of ± 1 count for tolerance. In many
2649 implementations, tolerance is specified as ± 1 count for these types of conversions. Because resolution is
2650 given in units of 1 count, tolerance and resolution may sometimes appear to equate to the same value.
2651 However, tolerance and resolution should not be misinterpreted as being the same thing.

2652 Lastly, in some cases PLDM Numeric Sensors will return values such as counts or other measurements
2653 that do not use a conversion process that can introduce errors in the reading. In this case, the tolerance is
2654 specified as ± 0 counts.

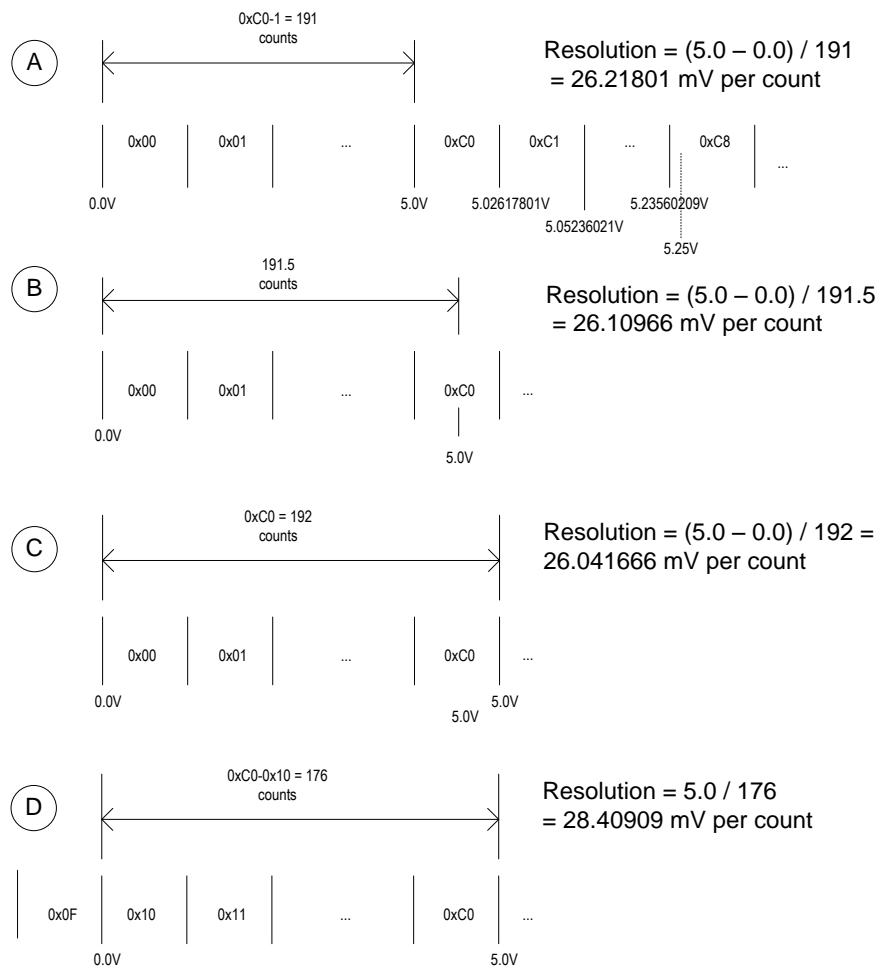
2655 **27.6.3 Accuracy, tolerance, and resolution relationship to thresholds**

2656 Accuracy, tolerance, and resolution must all be taken into account to generate a threshold that does not
2657 generate a "false positive" (a false indication of a failure). For example, if accuracy, tolerance, and
2658 resolution are not taken into account when calculating the threshold for a warning level, it is possible that
2659 an input could be assessed as being within the warning range when the input was actually near the limit
2660 of the normal range.

2661 A consequence of avoiding false positives is that for a particular range a value that is actually within the
2662 intended warning range can be assessed as being within the normal range. That is, false positives are
2663 avoided at the cost of having the possibility of 'false negatives'. However, in most implementations it is
2664 considered better to avoid the false alarms that false positives would cause. Whether to design thresholds
2665 to avoid false positives or false negatives is a choice of the system implementation.

2666 Because it is the more common case, the following examples describe how thresholds may be calculated
2667 to avoid false positives.

2668 EXAMPLE: An 8-bit A/D converter monitoring a 5.0 V nominal signal where the sensor has been designed such
2669 that the 5.0 V level corresponds to a reading of C0h and the 0.0 V level corresponds to a reading of
2670 00h (as shown by Figure 25A). Assume the converter implementation has a specified worst-case
2671 accuracy of $\pm 4\%$, and a tolerance of ± 1 count.



2672

2673

Figure 25 – Figuring resolution from the design

2674 For Figure 25A, this yields resolution, tolerance, and accuracy values as follows:

2675 Resolution

2676 $= 5.0 \text{ V} / (\text{C0h} - 1) = 26.17801 \text{ mV}$

2677 Accuracy

2678 $= \pm 4\%$ (given, from the design)

2679 Tolerance

2680 $= \pm 1 \text{ count (given)} = \pm 26.17801 \text{ mV}$

2681 Now, suppose it is necessary to calculate an upper critical threshold for the 5.0 V + 5% point (5.25 V)

2682 where this threshold will not produce "false positives" (falsely return 'critical') across the range of

2683 accuracy, tolerance, and resolution. The following example shows steps that can be used to calculate a

2684 threshold suitable for a PLDM Numeric Sensor:

2685 Step 1: Divide the target threshold value by the resolution to find how many counts correspond to
2686 5.25 V:

2687 $5.25 \text{ V} / 26.17801 \text{ mV} = 200.55 \text{ counts}$
2688 (which puts the 5.25 V point within the nominal range of reading 0xC8, as shown in
2689 Figure 25A)

2690 Step 2: Factor in the tolerance:

2691 **Important:** Because tolerance is specified as an error, a "+" count for tolerance means that
2692 the reading may be higher than it should be, and a "-" count means that the reading may be
2693 lower than it should be. To account for these errors, the "-" tolerance value should be added
2694 to upper thresholds, and the "+" tolerance value subtracted from lower thresholds. This is
2695 particularly important when the plus and minus tolerance values are different from one
2696 another.

2697 $200.55 + 1 = 201.55 \text{ counts}$

2698 Step 3: Account for the effect of accuracy:

2699 $201.55 * 1.04 = 209.612 \text{ counts}$

2700 Step 4: Round up (because an A/D converter cannot give a non-integer count)

2701 $209.612 \rightarrow 210 \text{ counts} = 0xD2$

2702 This yields a threshold value of 210, which corresponds to 5.497 V. This shows that even though a
2703 threshold of 5.25 V is being targeted, it is necessary to set the threshold to a value that, because of the
2704 effects of accuracy, tolerance, and resolution, could allow the actual monitored value to be as high as
2705 5.497 V in some implementations before a threshold match would be detected.

2706 The calculations for lower thresholds are the same, except that negative values for the accuracy,
2707 tolerance, and resolution are used.

2708 Figure 25 illustrates what to be aware of when deriving the values for resolution from an implementation.
2709 To get an accurate value for resolution, it is important to know whether the input values that correspond to
2710 a particular reading are given as values that are at the point of change (quantization point) between
2711 successive readings, are a nominal "center point" of a reading, or a combination of the two. (The
2712 difference in the resolution value between Figure 25A and Figure 25C is almost 0.5%. This shows that a
2713 nontrivial amount of error could be introduced if the implementer uses the wrong calculation point for its
2714 implementation).

2715 Lastly, area D in Figure 25 shows that offsets in the implementation also need to be taken into account.
2716 Offset adds a new first step to the threshold calculation:

2717 Step 0: Take the target threshold and subtract (or add, depending on the implementation) the D.C.
2718 offset value before calculating the counts for the threshold.

27.7 Numeric reading conversion formula

The following formula is used with data from the Numeric Sensor PDR to convert the corresponding PLDM Numeric Sensor's raw reading to the units specified in the Numeric Sensor PDR.

Reading Conversion formula: $Y = (m * X + B)$

Where:

Y = converted reading in Units

X = reading from sensor

m = resolution from PDR in Units

B = offset from PDR in Units

Units = sensor/effecter Units, based on the Units and auxUnits fields from the PDR for the numeric sensor

For example, a sensor with the following units, resolution, offset, and reading:

Reading = 0xBF

Units = Volts

Resolution: 26.17801 mV

Offset = -1.00 V

would have the following the converted reading:

$Y = (26.17801 * 10^{-3} \text{ V} * 0xBF + (-1.00 \text{ V})) = [(0.2617801 * 191) - 1.00] \text{ V} = 4.00 \text{ V}$

A full interpretation of the reading should also take tolerance and accuracy into account. For example, if the PDR indicates the following:

Accuracy: $\pm 4\%$

Tolerance: ± 1 count (given)

combined with the previous example, the full interpretation of the reading would be:

$(4.00 \text{ V} \pm 26.17801 \text{ mV}) \pm 4\%$

where $\pm 26.17801 \text{ mV}$ corresponds to the effect of a Tolerance of ± 1 count.

27.7.1 Rounding

Some precision may often be lost in the conversion of binary to decimal. For example, the previous conversion that was shown as 4.00 V actually calculates out to 3.99999991 V using the given value for the resolution, but the result was rounded up to 4.00. This raises a question about how much rounding should be applied, or how many digits of precision should be used for a converted value.

The number of digits of precision for the converted value can be based on the overall size of the binary number. For example, an eight-bit unsigned value has a range of 0 to 255, which is three decimal digits. Thus, rounding the converted reading to three significant digits is appropriate.

2752 **27.8 Numeric effector conversion formula**

2753 A reverse process from that used to convert a sensor reading is used to generate the raw value to be set
 2754 into a PLDM Numeric Effector. In this case, the formula is as follows:

2755 **Setting Conversion formula:** $X = \text{Round} [(Y - B) / m]$

2756 Where:

2757 X = integer setting value for the effector

2758 Y = target setting in Units

2759 m = resolution from PDR in Units

2760 B = offset from PDR in Units

2761 Round = rounding operation to round the value in [] to the nearest integer value

2762 Units = sensor/effector Units, based on the Units and auxUnits fields from the Numeric Effector
 2763 PDR

2764 **28 Platform Descriptor Record (PDR) formats**

2765 This clause defines the content and format of the PDRs that are used for supporting sensor monitoring
 2766 and control in PLDM.

2767 **28.1 Common PDR header format**

2768 All PDRs have a common, fixed format header followed by variable length record data. The size and
 2769 definition of the bytes within the PDR data field are specific to each PDR Type. Table 75 describes the
 2770 format of the common PDR header.

2771 The PDR data length can vary on a per record basis. It is generally recommended that the definition of
 2772 PDRs of a given type use a fixed length when practical.

2773 The header fields are not shown in the succeeding PDR format subclauses.

2774 **Table 75 – Common PDR header format**

Type	PDR fields
uint32	<p>recordHandle</p> <p>An opaque number that is used for accessing individual PDRs within a PDR Repository. The PDR Handle value is required to be unique for all PDRs within a PDR Repository. PDR Handle values are not required to be unique across PDR Types or across other PDRs in the system. See 26.2.3 for more information.</p> <p>special value: {0x0000_0000 = reserved }</p>
uint8	<p>PDRHeaderVersion</p> <p>This field is provided in case a future version of this specification requires a modification to the format of the PDR Header. Any PDR fields that follow this field are eligible for change.</p> <p>value: The value 0x01 shall be used as the PDRHeaderVersion for PDRs that are defined in this specification.</p>
uint8	<p>PDRType</p> <p>The type of the PDR. See 25.3 and 28.2.</p>

Type	PDR fields
uint16	<p>recordChangeNumber</p> <p>See 26.2.3 for more information.</p>
uint16	<p>dataLength</p> <p>The total number of PDR data bytes following this field.</p>

2775 **28.2 PDR type values**

2776 Table 76 lists the different types of PDRs defined in this document and the corresponding PDR Type
 2777 values used for those PDRs. Unspecified values are reserved for future definition by this specification.

2778

Table 76 – PDR Type Values

PDR type number	PDR type name	Reference
1	Terminus Locator PDR	See 28.3.
2	Numeric Sensor PDR	See 28.4.
3	Numeric Sensor Initialization PDR	See 28.5.
4	State Sensor PDR	See 28.6.
5	State Sensor Initialization PDR	See 28.7.
6	Sensor Auxiliary Names PDR	See 28.8.
7	OEM Unit PDR	See 28.9.
8	OEM State Set PDR	See 28.10.
9	Numeric Effector PDR	See 28.11.
10	Numeric Effector Initialization PDR	See 28.12.
11	State Effector PDR	See 28.13.
12	State Effector Initialization PDR	See 28.14.
13	Effector Auxiliary Names PDR	See 28.15.
14	Effector OEM Semantic PDR	See 28.16.
15	Entity Association PDR	See 28.17.
16	Entity Auxiliary Names PDR	See 28.18.
17	OEM Entity ID PDR	See 28.19.
18	Interrupt Association PDR	See 28.20.
19	PLDM Event Log PDR	See 28.21.
20	FRU Record Set PDR	See 28.22.
21	Compact Numeric Sensor PDR	See 28.25
22	Redfish Resource PDR	See 28.26
23	Redfish Entity Association PDR	See 28.27
24	Redfish Action PDR	See 28.28
25..125	Reserved for future use	
126	OEM Device PDR	See 28.23
127	OEM PDR	See 28.24.

2779 **28.3 Terminus Locator PDR**

2780 The Terminus Locator PDR provides information that associates a PLDMTerminusHandle with values that
 2781 uniquely identify the device or software that contains the PLDM terminus. Table 77 describes the format
 2782 of this PDR.

Table 77 – Terminus Locator PDR format

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus.
enum8	validity Indicates whether the PDR contains valid information for the terminus. This is also used as part of identifying (enumerating) which termini are present. See 12.5 for more information. value: { notValid, // The PDR should be ignored. valid // The PDR is valid. }
uint8	TID PLDM Terminus ID. This value is used to identify asynchronous messages from a given terminus.
uint16	containerID The containerID for the containing entity that holds this terminus. See 9.1 for more information.
enum8	terminusLocatorType value: { UID, MCTP_EID, SMBusRelative, // Used when the device has a fixed slave address and bus connection // that is relative to a device that is identified through a UID (for example, // if the terminus was an SMBus device on an add-in card and was // located on bus #3 of another device on that same add-in card that had // a UID) systemSoftware // Used when the terminus is a software or firmware agent that is running // under the host processors of the managed system }
uint8	terminusLocatorValueSize Size of the following terminusLocatorValue, in bytes. NOTE This helps facilitate backward compatibility in case terminusLocatorTypes get extended. The combination of terminusLocatorType and all fields of the terminusLocatorValue is persistent and unique for a given terminus in PLDM.
<i>terminusLocatorValue for terminusLocatorType = UID:</i>	
uint8	terminusInstance This field is used to differentiate between different PLDM termini if the device contains more than one PLDM terminus.

Type	Description
UUID	<p>deviceUUID</p> <p>Although using the UUID format, the value may not be universally unique among different platforms. For example, a device manufacturer could assign the same value to all the devices of a particular type that it manufactures, provided that only one instance of that device would be used within a given PLDM implementation. Similarly, a device manufacturer could manufacture a device that contains a set of UUIDs and provide a mechanism such as configuration pins or nonvolatile memory that would enable one UUID from the set to be selected when the device was integrated into the system. The value may also be derived from another UID or UUID, such as the unique ID for the device containing the terminus, a UUID for the overall system, and so on.</p> <p>A PLDM terminus that is identified using this type of ID must support the GetTerminusUUID command.</p>
<i>terminusLocatorValue for terminusLocatorType = MCTP_EID:</i>	
uint8	<p>EID</p> <p>A MCTP EID that is assigned to an MCTP Endpoint that provides the transport protocol termination for a PLDM terminus</p>
<i>terminusLocatorValue for terminusLocatorType = SMBusRelative</i>	
UUID	<p>UID</p> <p>A UID for the controller that owns the bus to which the device is connected. For more information, see the preceding description for "<i>terminusLocatorType = UID</i>".</p>
uint8	<p>busNumber</p> <p>A bus number for the bus to which the device is connected, relative to the controller that owns the bus.</p> <p>If the PLDM terminus is accessed through an MCTP Endpoint, the busNumber must be the port number used in the routing table for accessing the endpoint.</p>
uint8	<p>slaveAddress</p> <p>The SMBus or I²C slave address for the device that is providing the</p> <p>[7:1] - SMBus or I²C slave address value. [0] - 0b.</p>
<i>terminusLocatorValue for terminusLocatorType = systemSoftware</i>	
enum8	<p>softwareClass</p> <pre>{ unspecified, other, systemFirmware, OSloader, OS, CIMprovider, otherProvider, virtualMachineManager }</pre>
UUID	<p>UUID</p> <p>A UID for the software or instance of software that is acting as a PLDM terminus. This ID is required to be unique for the particular instance of software within the system that is providing or emulating a PLDM terminus within a single PLDM platform management subsystem implementation. For example, a software application running on a platform may emulate sensors for the purpose of generating events to be handled by PLDM. This piece of software can be assigned a fixed UUID by the software vendor that is used to identify it as a unique PLDM terminus. If multiple instances of that software could exist on the platform where each instance individually provides an emulation of a PLDM terminus, each instance must have a different UUID. Similarly, if a common piece of software implements multiple PLDM termini, each terminus must have a different UUID.</p>

2784 **28.4 Numeric Sensor PDR**

2785 The Numeric Sensor PDR is primarily used to describe the semantics of a PLDM Numeric Sensor to a
 2786 party such as a MAP. It also includes the factors that are used for converting raw sensor readings to
 2787 normalized units. The record also identifies the Entity that is being monitored by the sensor. Table 78
 2788 describes the format of this PDR.

2789 NOTE The Numeric Sensor PDR sensorID type in this clause has been changed in version 1.1.1 of this
 2790 specification from uint8 to uint16 to be consistent with GetSensorReading command.

2791

2792

Table 78 – Numeric Sensor PDR format

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus.
uint16	sensorID ID of the sensor relative to the given PLDM Terminus ID.
uint16	entityType The Type value for the entity that is associated with this sensor. See 9.1 for more information.
uint16	entityInstanceNumber The Instance Number for the entity that is associated with this sensor. See 9.1 for more information.
uint16	containerID The containerID for the containing entity that instantiates the entity that is measured by this sensor. See 9.1 for more information.
enum8	sensorInit Indicates whether the sensor requires initialization by the initializationAgent. value: { nolnit, // The Initialization Agent does not take any steps to initialize, enable, // or disable this particular sensor. useInitPDR, // The sensor has an associated Numeric Sensor Initialization PDR // that should be used to initialize the sensor. enableSensor, // Whenever the Initialization Agent runs, it will enable this sensor // using a SetNumericSensorEnable command to set the // operationalState. disableSensor. // Whenever the Initialization Agent runs, it will disable this sensor by // using the SetNumericSensorEnable command. }
bool8	sensorAuxiliaryNamesPDR true = sensor has a Sensor Auxiliary Names PDR false = sensor does not have an associated Sensor Auxiliary Names PDR

Type	Description
enum8	<p>baseUnit</p> <p>The base unit of the reading returned by this sensor. See 27.4 for more information. value: { see Table 74 }</p>
sint8	<p>unitModifier</p> <p>A power-of-10 multiplier for the baseUnit. See 27.4 for more information.</p>
enum8	<p>rateUnit</p> <p>value: { None, Per MicroSecond, Per MilliSecond, Per Second, Per Minute, Per Hour, Per Day, Per Week, Per Month, Per Year }</p>
uint8	<p>baseOEMUnitHandle</p> <p>This value is used to locate the corresponding PLDM OEM Unit PDR that defines the OEMUnit when the OEMUnit value is used for the baseUnit.</p>
enum8	<p>auxUnit</p> <p>The base unit of the reading returned by this sensor. See 27.4 for more information. value: { see Table 74 }</p>
sint8	<p>auxUnitModifier</p> <p>A power-of-10 multiplier for the auxUnit. See 27.4 for more information.</p>
enum8	<p>auxrateUnit</p> <p>value: { None, Per MicroSecond, Per MilliSecond, Per Second, Per Minute, Per Hour, Per Day, Per Week, Per Month, Per Year }</p>
enum8	<p>rel</p> <p>The relationship between the base unit and the auxiliary unit, as follows:</p> <p>value = { dividedBy, multipliedBy}</p> <p>dividedBy implies a "/" or "per" relationship, such as "per foot"</p> <p>multipliedBy implies a "*" operation, such as "foot*lbs (foot-lbs)"</p>
uint8	<p>auxOEMUnitHandle</p> <p>This value is used to locate the PLDM OEM Unit PDR that defines the OEMUnit if the OEMUnit value is used for the auxUnit.</p>
bool8	<p>isLinear</p> <p>Indicates whether a sensor is linear or dynamic in its range.</p> <p>For example, this value can be used by a MAP to populate the IsLinear attribute of CIM_NumericSensor.</p> <p>value: This field is set to "true" to show that a sensor is linear.</p>
enum8	<p>sensorDataSize</p> <p>The bit width and format of reading and threshold values that the sensor returns value: { uint8, sint8, uint16, sint16, uint32, sint32 }</p>
real32	<p>resolution</p> <p>The resolution of the sensor in Units (see 27.7).</p>

Type	Description
real32	<p>offset</p> <p>A constant value that is added in as part of the conversion process of converting a raw sensor reading to Units (see 27.7).</p>
uint16	<p>accuracy</p> <p>Given as a +/- percentage in 1/100ths of a % from 0.00 to 100.00. For example, the integer value 510 corresponds to ± 5.10%. See 27.6 for more information.</p>
uint8	<p>plusTolerance</p> <p>Tolerance is given in +/- counts of the reading value. It indicates a constant magnitude possible error in the quantization of an analog input to the sensor. It is possible that the tolerance could be asymmetric. The plusTolerance field provides the '+' value of the tolerance; the minusTolerance field provides the minus portion. For example, if plusTolerance is 0x02 and minusTolerance is 0x00, the tolerance is +2/-0 counts.</p> <p>See 27.6 for more information about how tolerance is defined and used.</p>
uint8	<p>minusTolerance</p> <p>Tolerance is given in +/- counts of the reading value. It indicates a constant magnitude possible error in the quantization of an analog input to the sensor. It is possible that the tolerance could be asymmetric. The plusTolerance field provides the '+' value of the tolerance; the minusTolerance field provides the minus portion. For example, if plusTolerance is 0x02 and minusTolerance is 0x00, the tolerance is +2/-0 counts.</p> <p>See 27.6 for more information about how tolerance is defined and used.</p>
uint8 sint8 uint16 sint16 uint32 sint32	<p>hysteresis</p> <p>The amount of hysteresis associated with the sensor thresholds, given in raw sensor counts. See 17.9 for more information. This value may be overridden if the sensor supports the SetSensorThresholds command.</p> <p>The size of this field is identified by sensorDataSize.</p> <p>value: 1 or greater</p> <p>special value: 0 = sensor does not use hysteresis</p>
bitfield8	<p>supportedThresholds</p> <p>For PLDM: bit field where bit position represents whether a given threshold is supported</p> <p>0x1b = threshold is supported</p> <p>0x0b = threshold is not supported</p> <p>[6:7] – reserved</p> <p>[5] – lowerThresholdFatal</p> <p>[4] – lowerThresholdCritical</p> <p>[3] – lowerThresholdWarning</p> <p>[2] – upperThresholdFatal</p> <p>[1] – upperThresholdCritical</p> <p>[0] – upperThresholdWarning</p>

Type	Description
bitfield8	<p>thresholdAndHysteresisVolatility</p> <p>Identifies under which conditions any threshold or hysteresis settings that were set through the SetSensorThresholds or SetSensorHysteresis command may be lost. The threshold values either return to default values or will require reinitialization through the Initialization Agent function.</p> <p>special value: 00000b = nonvolatile. The threshold settings retained indefinitely regardless of system state.</p> <p>[7:5] – reserved</p> <p>[4] – 1b = PLDM terminus returns to online condition</p> <p>[3] – 1b = System warm resets</p> <p>[2] – 1b = System hard resets</p> <p>[1] – 1b = PLDM subsystem power up</p> <p>[0] – 1b = Initialization Agent controller restart/update (initialize/reinitialize this sensor whenever the device that holds the Initialization Agent has been restarted or reinitialized)</p>
real32	<p>stateTransitionInterval</p> <p>How long the sensor device takes to do an enabledState change (worst case), in seconds.</p> <p>NOTE Because this is floating point format, fractional seconds can be represented. The real32 format also supports a value for "Unknown".</p>
real32	<p>updateInterval</p> <p>Polling or update interval in seconds expressed using a floating point number (generally corresponds to the CIM PollingInterval property)</p>
uint8 sint8 uint16 sint16 uint32 sint32	<p>maxReadable</p> <p>The maximum value that the sensor may return. The size of this field is given by the sensorDataSize field in this PDR.</p> <p>This number is given in the same format as the reading returned by the sensor. The conversion formula is used to convert this number to normalized units. See 27.7.</p>
uint8 sint8 uint16 sint16 uint32 sint32	<p>minReadable</p> <p>The minimum value that the sensor may return. The size of this field is given by the sensorDataSize field in this PDR.</p> <p>This number is given in the same format as the reading returned by the sensor. The conversion formula is used to convert this number to normalized units. See 27.7.</p>
enum8	<p>rangeFieldFormat</p> <p>Indicates the format used for the following nominalValue, normalMax, normalMin, criticalHigh, criticalLow, fatalHigh, and fatalLow fields.</p> <p>NOTE The "warningHigh" and "warningLow" fields are not listed in this field. This is an error in the original specification and will be corrected in the next major release of this specification. The compact PDR provides these fields if required by the implementer.</p> <p>value: { uint8, sint8, uint16, sint16, uint32, sint32, real32 }</p>

Type	Description
bitfield8	<p>rangeFieldSupport</p> <p>Indicates which of the fields that identify the operating ranges of the parameter monitored by the sensor are supported. (This bitfield indicates whether the following nominalValue, normalMax, and so on, fields contain valid range values.)</p> <p>NOTE The “warningHigh” and “warningLow” fields are not listed in this field. The industry practice assumes that warningHigh and warningLow are always supported. This is an error in the original specification and will be corrected in the next major release of this specification. The compact PDR provides these fields if required by the implementer.</p> <p>[7] – reserved</p> <p>[6] – 1b = fatalLow field supported</p> <p>[5] – 1b = fatalHigh field supported</p> <p>[4] – 1b = criticalLow field supported</p> <p>[3] – 1b = criticalHigh field supported</p> <p>[2] – 1b = normalMin field supported</p> <p>[1] – 1b = normalMax field supported</p> <p>[0] – 1b = nominalValue field supported</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>nominalValue</p> <p>This value presents the nominal value for the parameter that is monitored by the sensor. The size of this field is given by the rangeFieldFormat field in this PDR. This value is given directly in the specified units without the use of any conversion formula.</p> <p>For example, if the units are millivolts and the nominalValue is 5000, the nominalValue corresponds to 5000 mV, or 5.000 V. It is possible that the nominal value could be some fraction of the given units for the sensor (for example, if the units are volts and the nominal value is 2.5 V). For this reason, the nominalValue can be expressed using a real32.</p> <p>The value is defined as the nominal value for what is being monitored. Thus, nominalValue is not required to match a value that can be returned as a reading by the sensor implementation. For example, if the nominal value for a given monitored voltage is 5.00 V, the nominalValue would typically be reported as 5.00 V even though the closest reading the sensor implementation may be able to return is 5.05 V.</p> <p>A common use of the nominalValue is as a source of part of an identifying 'name' for a sensor. For example, it is common for voltage sensors to be identified by their nominal reading. So, a sensor with a nominal reading of +5.00 V would be referred to as a "+5 V sensor", while one with a nominal reading of +3.3 V would be referred to as a "+3.3 V sensor". The definition of nominalValue in the PDR supports this usage. An application that uses or displays this value will typically elect to round the value to some number of significant digits using an algorithm based on the resolution of the sensor. For example, if the nominalValue is given as a real32 as 2.50000 V, but the resolution of the sensor is 0.05 V, the nominalValue displayed would typically be rounded as 2.50 V.</p> <p>It is possible that a given sensor may not be considered as having a nominal reading, in which case this field should be ignored. For example, a numeric sensor that tracks a count or size of some parameter may not be considered as having a nominal reading depending on its application.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>normalMax</p> <p>The upper limit of the normal operating range for the parameter that is monitored by the numeric sensor. The monitored parameter is considered to be operating outside of normal range when this value is exceeded. For example, if a monitored voltage of a component is specified in its data sheet to have a normal maximum operating range of 4.75 to 5.25 V, this value would be set to 5.25 (assuming the units in the PDR are for “volts”). This value is given directly in the specified units without the use of any conversion formula. This value is used together with normalMin to indicate the normal operating range for the sensor.</p>

Type	Description
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>normalMin</p> <p>The lower limit of the normal operating range for the parameter that is monitored by the numeric sensor. Sensor thresholds are typically set for a value that is lower than normalMin to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an “out-of-range” event state. This value is given directly in the specified units without the use of any conversion formula.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>warningHigh</p> <p>A warning condition that occurs when the monitored value is <i>greater than</i> the value reported by warningHigh. In many implementations, this value may be the same value as normalMax. Sensor thresholds that may be derived from this value are typically set for a value that is higher than warningHigh to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>warningLow</p> <p>A warning condition that occurs when the monitored value is <i>less than or equal to</i> the value reported by warningLow. In many implementations, this value may be the same value as normalMin. Sensor thresholds that may be derived from this value are typically set for a value that is lower than warningLow to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>criticalHigh</p> <p>A critical condition that occurs when the monitored value is <i>greater than or equal to</i> the value reported by criticalHigh. In some implementations, this value may be the same value as normalMax. Sensor thresholds that may be derived from this value are typically set for a value that is higher than criticalHigh to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>criticalLow</p> <p>A critical condition that occurs when the monitored value is <i>less than</i> the value reported by criticalLow. In some implementations, this value may be the same value as normalMin. Sensor thresholds that may be derived from this value are typically set for a value that is lower than criticalLow to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>fatalHigh</p> <p>A fatal condition that occurs when the monitored value is <i>greater than</i> the value reported by fatalHigh. In many implementations, this value may be the same value as normalMax. Sensor thresholds that may be derived from this value are typically set for a value that is higher than fatalHigh to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>fatalLow</p> <p>A fatal condition that occurs when the monitored value is <i>less than</i> the value reported by fatalLow. In many implementations, this value may be the same value as normalMin. Sensor thresholds that may be derived from this value are typically set for a value that is lower than fatalLow to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.</p>

2793 **28.5 Numeric Sensor Initialization PDR**

2794 The Numeric Sensor Initialization PDR is used when a PLDM Numeric Sensor requires initialization by a
 2795 PLDM Initialization Agent. Table 79 describes the format of this PDR.

2796 **Table 79 – Numeric Sensor Initialization PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus
uint16	sensorID ID of the sensor relative to the given PLDM Terminus ID
bitfield8	initConditions Identifies under which conditions the Initialization Agent must initialize or reinitialize this sensor [7:5] – reserved [4] – 1b = PLDM terminus returns to online condition [3] – 1b = System warm resets [2] – 1b = System hard resets [1] – 1b = PLDM subsystem power up [0] – 1b = Initialization Agent controller restart/update (initialize/reinitialize this sensor whenever the device that holds the Initialization Agent has been restarted or reinitialized)
enum8	sensorEnable The operational state that the sensor is to be left in after it has been initialized. This state is written to the sensor sensorOperationalState using the SetNumericSensorEnable command. special value: { 0xFF = do not change the sensorOperationalState }
bitfield8	thresholdInitMask Indicates which thresholds should be initialized NOTE Be careful to match the bit up with the correct threshold. [7:6] – reserved [5] – 1b = initialize lowerThresholdFatal threshold [4] – 1b = initialize lowerThresholdCritical threshold [3] – 1b = initialize lowerThresholdWarning threshold [2] – 1b = initialize upperThresholdFatal threshold [1] – 1b = initialize upperThresholdCritical threshold [0] – 1b = initialize upperThresholdWarning threshold
enum8	sensorDataSize The bit width of reading and threshold values that the sensor returns value: { uint8, sint8, uint16, sint16, uint32, sint32 }

Type	Description
uint8 sint8 uint16 sint16 uint32 sint32	upperThresholdWarning This value is given in raw units for the sensor. The size of this field is given by the sensorDataSize field in this PDR.
uint8 sint8 uint16 sint16 uint32 sint32	upperThresholdCritical This value is given in raw units for the sensor. The size of this field is given by the sensorDataSize field in this PDR.
uint8 sint8 uint16 sint16 uint32 sint32	upperThresholdFatal This value is given in raw units for the sensor. The size of this field is given by the sensorDataSize field in this PDR.
uint8 sint8 uint16 sint16 uint32 sint32	lowerThresholdWarning This value is given in raw units for the sensor. The size of this field is given by the sensorDataSize field in this PDR.
uint8 sint8 uint16 sint16 uint32 sint32	lowerThresholdCritical This value is given in raw units for the sensor. The size of this field is given by the sensorDataSize field in this PDR.
uint8 sint8 uint16 sint16 uint32 sint32	lowerThresholdFatal This value is given in raw units for the sensor. The size of this field is given by the sensorDataSize field in this PDR.

2797 **28.6 State Sensor PDR**

2798 The State Sensor PDR provides the sensorID for a composite state sensor within a PLDM terminus and
 2799 the number of sensors, and the state set and the possible state values for each sensor that is accessed
 2800 through the given sensorID. The record also identifies the entity that is being monitored by the sensor.
 2801 Only one set of fields exists for the entity identification information. Therefore, all sensors in this record
 2802 must be associated with the same entity. Table 80 describes the format of this PDR.

2803 **Table 80 – State Sensor PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus
uint16	sensorID ID of the sensor relative to the given PLDM Terminus ID
uint16	entityType The Type value for the entity that is associated with this sensor. See 9.1 for more information.
uint16	entityInstanceNumber The Instance Number for the entity that is associated with this sensor. See 9.1 for more information.

Type	Description
uint16	<p>containerID</p> <p>The containerID for the containing entity that instantiates the entity that is measured by this sensor. See 9.1 for more information.</p>
enum8	<p>sensorInit</p> <p>Indicates whether the sensor requires initialization by the initializationAgent.</p> <p>value: { nolnit, // The Initialization Agent does not take any steps to initialize, // enable, or disable this particular sensor.</p> <p> useInitPDR, // The sensor has an associated State Sensor Initialization PDR // that should be used to initialize the sensor.</p> <p> enableSensor, // When the Initialization Agent runs, it enables this sensor using // a SetStateSensorEnables command to set the // operationalState.</p> <p> disableSensor. // When the Initialization Agent runs, it disables this sensor using // the SetStateSensorEnables command.</p> <p>}</p>
bool8	<p>sensorAuxiliaryNamesPDR</p> <p>true = sensor has a Sensor Auxiliary Names PDR</p> <p>false = sensor does not have an associated Sensor Auxiliary Names PDR</p>
uint8	<p>compositeSensorCount</p> <p>The number of state sensors in the terminus that are accessed under the sensorID given in this PDR</p> <p>value: 0x01 to 0x08</p>
var	<p>possibleStates</p> <p>One instance of State Sensor Possible States Fields (see Table 81) for each sensor in the PLDM State Sensor, up to sensorCount.</p>

2804

Table 81 – State Sensor possible states fields format

Type	Description
uint16	<p>stateSetID</p> <p>A numeric value that identifies the PLDM State Set that is used with this sensor</p>
uint8	<p>possibleStatesSize</p> <p>The number of bytes (M) in the following possibleStates bitfield</p> <p>value: 0x01 to 0x20</p> <p>special value : 0x00 can be used to indicate a sensor that is unavailable or disabled from use and should be ignored when accessing the parent compositeSensor through PLDM.</p>

Type	Description
bitfield8 x M	<p>possibleStates [subset of the State Set that is supported]</p> <p>A variable length bitfield consisting of one or more bytes, based on the size of the stateSet. If stateSetSize is nonzero, possibleStates consists of one or more 8-bit fields where X = 0 for the first field, X = 1 for the second field (if any), and so on, up to M fields as required by the size of the largest value in the state set.</p> <p>For example, if the largest value in the State Set is 7 or less, this is a one-byte bitfield. If the largest value in the State Set is 15 or less, this is a two-byte bitfield, and so on.</p> <p>The value 0b is also used when there is no state set value that corresponds to the corresponding bit position. For example, if a state set has a maximum value of 5, bits [6] and [7] are unused and shall be set to 0b.</p> <p>[7] – 1b = The state that corresponds to value X*8+7 in the state set is supported. 0b = The state that corresponds to value X*8+7 in the state set is not supported.</p> <p>...</p> <p>[2] – 1b = The state that corresponds to value X*8+2 in the state set is supported. 0b = The state that corresponds to value X*8+2 in the state set is not supported.</p> <p>[1] – 1b = The state that corresponds to value X*8+1 in the state set is supported. 0b = The state that corresponds to value X*8+1 in the state set is not supported.</p> <p>[0] – 1b = The state that corresponds to value X*8+0 in the state set is supported. 0b = The state that corresponds to value X*8+0 in the state set is not supported.</p>

2805 **28.7 State Sensor Initialization PDR**

2806 The State Sensor Initialization PDR contains values that direct the Initialization Agent's initialization of a
 2807 particular PLDM Single or Composite State Sensor. This action includes enabling or disabling PLDM
 2808 Event Message generation for individual sensors within the PLDM Composite State Sensor and directing
 2809 whether a particular sensor will assess an event if the initialization state value does not match the present
 2810 state of the sensor.

2811 The PDR always has eight state values (stateValue0 through stateValue7). Dummy values must be used
 2812 (0x00 is recommended) if the implementation does not have a sensor that corresponds to a particular
 2813 offset. Table 82 describes the format of the PDR.

2814 **Table 82 – State Sensor Initialization PDR format**

Type	Description
–	<p>commonHeader</p> <p>See 28.1.</p>
uint16	<p>PLDMTerminusHandle</p> <p>A handle that identifies PDRs that belong to a particular PLDM terminus</p>
uint16	<p>sensorID</p> <p>ID of the sensor relative to the given PLDM terminus</p>

Type	Description
bitfield8	<p>initConditions</p> <p>Identifies under which conditions the Initialization Agent must initialize or reinitialize these sensors</p> <p>The initConditions are shared across all sensors that are identified as requiring initialization through the sensorInitMask field. If some sensors require different initialization conditions, a separate PLDM Composite State Sensor Initialization PDR must be used for those sensors.</p> <p>[7:5] – reserved</p> <p>[4] – 1b = PLDM terminus returns to online condition</p> <p>[3] – 1b = System warm resets</p> <p>[2] – 1b = System hard resets</p> <p>[1] – 1b = PLDM subsystem power up</p> <p>[0] – 1b = Initialization Agent controller restart/update (initialize/reinitialize this sensor whenever the device that holds the Initialization Agent has been restarted or reinitialized)</p>
enum8	<p>sensorEnable</p> <p>The operational state of the overall composite state sensor after it has been initialized. This state is written to the sensorOperationalState of each sensor that is identified for initialization through the sensorInitMask field of this PDR using the SetStateSensorEnables command.</p> <p>special value: {0xFF = do not set the sensorOperationalStates}</p>
bitfield8	<p>sensorInitMask</p> <p>Identifies which sensors within the composite state sensor require initialization</p> <p>[7] – 1b = state sensor at offset 7 requires initialization 0b = state sensor at offset 7 does not require initialization</p> <p>[6] – 1b = state sensor at offset 6 requires initialization 0b = state sensor at offset 6 does not require initialization</p> <p>...</p> <p>[2] – 1b = state sensor at offset 2 requires initialization 0b = state sensor at offset 2 does not require initialization</p> <p>[1] – 1b = state sensor at offset 1 requires initialization 0b = state sensor at offset 1 does not require initialization</p> <p>[0] – 1b = state sensor at offset 0 requires initialization 0b = state sensor at offset 0 does not require initialization</p>

Type	Description
bitfield8	<p>sensorOpStateEventEnableMask</p> <p>Identifies which sensors within the composite state sensor should have their operational state event message generation enabled after initialization</p> <p>[7] – 1b = enable event message generator for state sensor at offset 7 0b = disable event message generator for state sensor at offset 7</p> <p>[6] – 1b = enable event message generator for state sensor at offset 6 0b = disable event message generator for state sensor at offset 6</p> <p>...</p> <p>[2] – 1b = enable event message generator for state sensor at offset 2 0b = disable event message generator for state sensor at offset 2</p> <p>[1] – 1b = enable event message generator for state sensor at offset 1 0b = disable event message generator for state sensor at offset 1</p> <p>[0] – 1b = enable event message generator for state sensor at offset 0 0b = disable event message generator for state sensor at offset 0</p>
bitfield8	<p>sensorStateEventEnableMask</p> <p>Identifies which sensors within the composite state sensor should have their state event message generation enabled after initialization</p> <p>[7] – 1b = enable event message generator for state sensor at offset 7 0b = disable event message generator for state sensor at offset 7</p> <p>[6] – 1b = enable event message generator for state sensor at offset 6 0b = disable event message generator for state sensor at offset 6</p> <p>...</p> <p>[2] – 1b = enable event message generator for state sensor at offset 2 0b = disable event message generator for state sensor at offset 2</p> <p>[1] – 1b = enable event message generator for state sensor at offset 1 0b = disable event message generator for state sensor at offset 1</p> <p>[0] – 1b = enable event message generator for state sensor at offset 0 0b = disable event message generator for state sensor at offset 0</p>
bitfield8	<p>sensorEventRearm</p> <p>Directs the sensor to assess an event if the initialization stateValue does not match the present state, or to accept the initialization stateValue as its initial state and ignore any prior state</p> <p>sensorEventRearm value:</p> <p>1b = trigger an event if the initialization stateValue does not match the present state 0b = accept the initialization stateValue as the present state</p> <p>[7] – sensorEventRearm value for the state sensor at offset 7</p> <p>[6] – sensorEventRearm value for the state sensor at offset 6</p> <p>...</p> <p>[2] – sensorEventRearm value for the state sensor at offset 2</p> <p>[1] – sensorEventRearm value for the state sensor at offset 1</p> <p>[0] – sensorEventRearm value for the state sensor at offset 0</p>

Type	Description
uint8	stateValue0 State value to write to sensor offset 0 for initialization special value: Use 0x00 as a placeholder value for sensors that do not require initialization.
uint8	stateValue1 State value to write to sensor offset 1 for initialization special value: Use 0x00 as a placeholder value for sensors that do not require initialization.
uint8	stateValue2 State value to write to sensor offset 2 for initialization special value: Use 0x00 as a placeholder value for sensors that do not require initialization.
	...
uint8	stateValue6 State value to write to sensor offset 14 for initialization special value: Use 0x00 as a placeholder value for sensors that do not require initialization.
uint8	stateValue7 State value to write to sensor offset 15 for initialization special value: Use 0x00 as a placeholder value for sensors that do not require initialization.

2815 **28.8 Sensor Auxiliary Names PDR**

2816 The Sensor Auxiliary Names PDR may be used to provide optional information that names the sensor.
 2817 This record may be used for a single numeric or state sensor, or multiple sensors if the sensor is a
 2818 composite state sensor.

2819 The nameLanguageTag field can be used to identify the language (such as French, Italian, or English)
 2820 that is associated with the particular sensorName. Table 83 describes the format of this PDR.

2821 **Table 83 – Sensor Auxiliary Names PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus
uint16	sensorID ID of the sensor relative to the given PLDM terminus

Type	Description
uint8	<p>sensorCount [1..M]</p> <p>For each sensor x in sensorCount, there can be 1..nameStringCount[x] strings, where each set of strings corresponds to a sensor in a composite sensor. The record must be populated sequentially starting from 1 regardless of whether a sensor requires auxiliary names. Thus, each entry has at least one byte (the nameStringCount). Sensors that have offsets that are greater than sensorCount are treated as if they have no auxiliary names.</p> <p>For example, if a composite sensor contains four sensors and only the third sensor requires an auxiliary name, the sensorCount can be 3 and the nameStringCount for the first two sets of sensor name information is 0.</p>
uint8	<p>nameStringCount</p> <p>Number of following pairs [0..N] of nameLanguageTag + sensorName fields for sensor[1].</p>
strASCII	<p>nameLanguageTag [1]</p> <p>This field is absent if nameStringCount = 0.</p> <p>A null-terminated ISO646 ASCII string that holds a language tag, per RFC4646, that identifies the primary language in which the sensorName was defined (for example, "en" for English, "zh-cmn-Hans" for simplified Mandarin Chinese, and so on). This field may be used to help select which string to use when multiple character encodings for the sensorName are provided.</p> <p>special value: null string = 0x0000 = unspecified</p>
strUTF-16BE	<p>sensorName [1]</p> <p>This field is absent if nameStringCount = 0.</p> <p>A null-terminated unicode string for the auxiliary name of the sensor</p> <p>special value: null string = 0x0000 = name not provided</p>
...	...
strASCII	nameLanguageTag [N]
strUTF-16BE	sensorName [N]

2822 **28.9 OEM Unit PDR**

2823 The OEM Unit PDR is used to define one or more strings that are used as the name for an OEM Unit
 2824 used for PLDM sensors or effecters. The OEM Unit is defined relative to the given Vendor ID and for a
 2825 given terminus. The OEMUnitHandle value is required to be unique among all OEM Unit PDRs within a
 2826 PDR Repository. The OEMUnitHandle value is not required to be unique across PDR Repositories.

2827 The record also includes a vendor-defined OEMUnitID value that identifies different types of OEM Units
 2828 from the given vendor.

2829 The record allows the unit name to be specified using multiple character sets. The unitLanguageTag can
 2830 be used to identify the language that is associated with the particular unitName (for example, whether the
 2831 unitName is in French, Italian, English, and so on). Table 84 describes the format of this PDR.

2832 **Table 84 – OEM Unit PDR format**

Type	Description
–	<p>commonHeader</p> <p>See 28.1.</p>

Type	Description
uint16	PLDMTerminusHandle The terminus that originated this PDR
uint8	OEMUnitHandle An opaque number that is used to identify different OEM Units PDRs
uint32	vendorIANA The IANA Enterprise Number for the vendor that is defining the OEM Sensor Unit
uint8	OEMUnitID A search field for the FindPDR command. This number is assigned by the vendor and provides a numeric ID for the vendor-defined Unit. This value can be used by the vendor to provide a constant ID that always identifies a particular Unit definition from that vendor.
uint8	stringCount The number 1..N of unitLanguageTag and unitName field pairs that follow this field
strASCII	unitLanguageTag[1] A null-terminated ISO646 ASCII string that holds a language tag, per RFC4646 , that identifies the primary language in which the unitName was defined (for example, "en" for English, "zh-cmn-Hans" for simplified Mandarin Chinese, and so on). This field may be used to help select which string to use when multiple character encodings for the unitName are provided. special value: null string = unspecified
strUTF-16BE	unitName[1] A null-terminated unicode string that contains the name of the OEM Sensor Unit
...	...
strASCII	unitLanguageTag[N]
strUTF-16BE	unitName[N]

2833 28.10 OEM State Set PDR

2834 The OEM State Set PDR is used to identify the vendor and OEM State Set ID value when the stateSetID
 2835 is treated as an OEMStateSetIDHandle. The PDR can also optionally be used to provide names for the
 2836 different OEM-defined states. Each different state can be assigned a name in one or more languages. A
 2837 contiguous range of state values can also be assigned a single set of names. It is also possible for the
 2838 PDR to provide a "hint" to help an entity such as a MAP decide how to treat state values that are not
 2839 explicitly specified in the PDR. The OEM State Set PDR is applicable to OEM State Sets for both sensors
 2840 and effecters.

2841 Depending on what range the stateSetID value falls in, the stateSetID value in a PDR, such as the PLDM
 2842 State Sensor PDR, either identifies the state set number for a particular state set defined in [DSP0249](#) or
 2843 is a value that is interpreted as an OEMStateSetIDHandle. The OEMStateSetIDHandle value is used to
 2844 form an association with a particular PLDMOEMStateSetPDR within the PDR Repository.
 2845 OEMStateSetIDHandle values are thus required to be unique for each different PLDM OEM State Set
 2846 PDR within a given PDR Repository.

2847 The following example describes the steps that could be taken to interpret the state value information
 2848 from an event message that originated from a PLDM State Sensor. This includes showing the difference
 2849 between using one of the standard state set numbers and an OEM State Set number.

2850 1) A PLDM Event Message is received from a state sensor.

- 2851 2) The TID, sensorID, sensorOffset, and state values (that is, eventState and previousEventState)
 2852 are read from the message.
- 2853 3) The TID is used to look up the Terminus Locator Record and obtain the PLDMTerminusHandle
 2854 value that is associated with the TID.
- 2855 4) PLDMTerminusHandle and sensorID values are used to look up the PLDM State Sensor PDR
 2856 for the sensor.
- 2857 5) The Sensor Offset is used to get the stateSetID from the PLDM State Sensor PDR. If the
 2858 stateSetID is in the range of standard IDs, the meaning of the state value is given according to
 2859 the stateSetID defined by the state set identified in [DSP0249](#).
- 2860 6) Otherwise the stateSetID from the PLDM State Sensor PDR is used as an
 2861 OEMStateSetIDHandle to look up the OEM State Set PDR that defines the OEM State Set. The
 2862 PDR identifies the OEM that defined the state set and provides the OEM-specified State Set
 2863 number (OEMStateSetID) for the state set. The state value from the event message can be
 2864 used to locate the OEM State Value Record in the PLDM OEM State Set PDR that provides a
 2865 name string for the particular OEM-defined state.

2866 Table 85 describes the format of the PDR.

2867 **Table 85 – OEM State Set PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle The terminus that originated this PDR
uint16	OEMStateSetIDHandle An OEM State Set within this PDR Repository. The value is taken from the range of OEMStateSet numbers defined in DSP0249 . This value is used in place of standard State Set ID numbers in the PDR for the sensor. When a value in the OEM State Set range is used as the State Set ID in a PDR, it indicates that the corresponding PLDM OEM State Set PDR should be referenced in order to get the OEM identification and definition for the OEM State Set.
uint32	vendorIANA The IANA Enterprise Number for the vendor that is defining the OEM State Set given in this PDR
uint16	OEMStateSetID A number, assigned by the vendor, that provides a numeric ID for the vendor-defined state set. The vendor can use this value to provide a constant ID that always identifies a particular state set from that vendor. The value shall be in the range defined for OEM State Set numbers defined in DSP0249 .
enum8	unspecifiedValueHint This field can be used to provide a hint to a higher level entity, such as a MAP, regarding how OEM state values should be treated if they are not explicitly covered by the OEMStateValueRecords field. value: { treatAsUnspecified, treatAsError }

Type	Description
uint8	stateCount The number of OEM State Value Records following this field in the PDR. Records shall be stored starting from the lowest stateValue to the highest.
variable	OEMStateValueRecord Zero or more OEM State Value Records as specified by the stateCount field. See Table 86.

2868

Table 86 – OEM State Value Record format

Type	Description
uint8	minStateValue The lowest state enumeration value that corresponds to the definition given in this OEM State Value Record instance.
uint8	maxStateValue The highest state enumeration value that corresponds to the definition given in this OEM State Value Record instance. State value ranges are not allowed to overlap. If maxStateValue = minStateValue, the following strings apply only to a single state. If maxStateValue > minStateValue, the following strings apply to state values in the range from minStateValue through maxStateValue.
uint8	stringCount The number 1..N of stateLanguageTag and stateName field pairs that follow this field.
strASCII	stateLanguageTag[1] A null-terminated ISO646 ASCII string that holds a language tag, per RFC4646 , that identifies the primary language in which the stateName was defined (for example, "en" for English, "zh-cmn-Hans" for simplified Mandarin Chinese, and so on). This field may be used to help select which string to use when multiple character encodings for the stateName are provided. special value: null string = unspecified
strUTF-16BE	stateName[1] A null-terminated unicode string that contains the name for the state
...	...
strASCII	stateLanguageTag[N]
strUTF-16BE	stateName[N]

2869

28.11 Numeric Effector PDR

2870 The Numeric Effector PDR is used to describe the semantics of a PLDM Numeric Effector to a party such
 2871 as a MAP. It also includes the factors that are used for converting raw sensor readings to normalized
 2872 units. The PDR also identifies the entity on which the effector is operating. Table 87 describes the format
 2873 of the PDR.

2874 NOTE The Numeric Effector PDR effectorID type in this clause has been changed in version 1.1.1 of this
 2875 specification from uint8 to uint16 to be consistent with SetNumericEffectorEnable command.

2876

2877

Table 87 – Numeric Effector PDR format

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus
uint16	effectorID ID of the effector relative to the given PLDM Terminus ID.
uint16	entityType The Type value for the entity that is associated with this effector. See 9.1 for more information.
uint16	entityInstanceNumber The Instance Number for the entity that is associated with this effector. See 9.1 for more information.
uint16	containerID The containerID for the containing entity that is associated with this effector. See 9.1 for more information.
uint16	effectorSemanticID This field either identifies a PLDM-defined effector semantic or provides an OEMEffectorSemanticHandle value, depending on what range the value falls in. If the effectorSemanticID field is set to a value in the OEM range, this value does not directly identify a particular vendor-defined semantic but instead is interpreted as an OEMEffectorSemanticHandle that can be used to locate an OEM Effector Semantic PDR that identifies the vendor and provides optional name information for the semantic. See DSP0249 for the definition of Effector Semantic ID values and ranges, and 21.3 for more information. special value: {0x0000 = unspecified }
enum8	effectorInit value: { nolnit, // The Initialization Agent does not take any steps to initialize, // enable, or disable this particular sensor. useInitPDR, // The sensor has an associated Numeric Effector Initialization // PDR that should be used to initialize the sensor. enableEffector, // When the Initialization Agent runs, it enables this effector using // a SetNumericEffectorEnable command to set the // operationalState. disableEffector // When the Initialization Agent runs, it disables this effector using // the SetNumericEffectorEnable command. }
bool8	effectorAuxiliaryNames PDR true = effector has an Effector Auxiliary Names PDR false = effector does not have an associated Effector Auxiliary Names PDR
enum8	baseUnit The base unit of the reading returned by this effector. See 27.1 for more information. value: { see Table 74 }

Type	Description
sint8	<p>unitModifier</p> <p>A power-of-10 multiplier for the baseUnit. See 27.1 for more information.</p>
enum8	<p>rateUnit</p> <p>value: { None, Per MicroSecond, Per MilliSecond, Per Second, Per Minute, Per Hour, Per Day, Per Week, Per Month, Per Year }</p>
uint8	<p>baseOEMUnitHandle</p> <p>This value is used to locate the PLDM OEM Unit PDR that defines the OEMUnit if the OEMUnit value is used for the baseUnit.</p>
enum8	<p>auxUnit</p> <p>The base unit of the reading returned by this effector. See 27.2 for more information.</p> <p>value: { see Table 74 }</p>
sint8	<p>auxUnitModifier</p> <p>A power-of-10 multiplier for the auxUnit. See 27.2 for more information.</p>
enum8	<p>auxrateUnit</p> <p>value: { None, Per MicroSecond, Per MilliSecond, Per Second, Per Minute, Per Hour, Per Day, Per Week, Per Month, Per Year }</p>
uint8	<p>auxOEMUnitHandle</p> <p>This value is used to locate the PLDM OEM Unit PDR that defines the OEMUnit if the OEMUnit value is used for the auxUnit.</p>
bool8	<p>isLinear</p> <p>Indicates whether a sensor is linear or dynamic in its range.</p> <p>For example, this value is used to provide information that can be used by a MAP to populate the IsLinear attribute of CIM_NumericSensor.</p> <p>value: This field is set to "true" to show that a sensor is linear.</p>
enum8	<p>effectorDataSize</p> <p>The bit width and format of reading and threshold values that the effector returns</p> <p>value: { uint8, sint8, uint16, sint16, uint32, sint32 }</p>
real32	<p>resolution</p> <p>The resolution of the effector in Units (see 27.7)</p>
real32	<p>offset</p> <p>A constant value that is added as part of the conversion process of converting a raw effector reading to Units (see 27.7).</p>
uint16	<p>accuracy</p> <p>Given as a +/- percentage in 1/100ths of a % from 0.00 to 100.00. For example, the integer value 510 corresponds to ± 5.10%. See 27.6 for more information.</p>

Type	Description
uint8	<p>plusTolerance</p> <p>Tolerance is given in +/- counts of the setting value. It indicates a constant magnitude possible error in the generation of an analog output from an effector. It is possible that the tolerance could be asymmetric. The plusTolerance field provides the “+” value of the tolerance; the minusTolerance field provides the minus portion. For example, if plusTolerance is 0x02 and minusTolerance is 0x00, the tolerance is +2/-0 counts.</p> <p>See 27.6 for more information about how tolerance is defined and used.</p>
uint8	<p>minusTolerance</p> <p>Tolerance is given in +/- counts of the setting value. It indicates a constant magnitude possible error in the generation of an analog input from an effector. It is possible that the tolerance could be asymmetric. The plusTolerance field provides the “+” value of the tolerance; the minusTolerance field provides the minus portion. For example, if plusTolerance is 0x02 and minusTolerance is 0x00, the tolerance is +2/-0 counts.</p> <p>See 27.6 for more information about how tolerance is defined and used.</p>
real32	<p>stateTransitionInterval</p> <p>The length of time the effector takes to do an enabledState change (worst case), in seconds</p> <p>NOTE Because this is floating point format, fractional seconds can be represented. The real32 format also supports a value for "Unknown".</p>
real32	<p>TransitionInterval</p> <p>The length of time the effector takes to have a setting change take effect (worst case), in seconds.</p>
uint8 sint8 uint16 sint16 uint32 sint32	<p>maxSettable</p> <p>The maximum legal setting value that the effector accepts. The size of this field is given by the effectorDataSize field in this PDR.</p> <p>This number is given in the same format as the reading returned by the effector. The conversion formula is used to convert this number to normalized units. See definition in 27.1.</p>
uint8 sint8 uint16 sint16 uint32 sint32	<p>minSettable</p> <p>The minimum legal setting value that the effector accepts. The size of this field is given by the effectorDataSize field in this PDR.</p> <p>This number is given in the same format as the reading returned by the effector. The conversion formula is used to convert this number to normalized units. See definition in 27.1.</p>
enum8	<p>rangeFieldFormat</p> <p>Indicates the format used for the following nominalValue, normalMax, and normalMin fields.</p> <p>value: { uint8, sint8, sint16, uint32, sint32, real32 }</p>
Bitfield8	<p>rangeFieldSupport</p> <p>This field indicates which of the fields that identify the operating ranges of the parameter set by the effector are supported. (This bitfield indicates whether the following nominalValue, normalMax, and so on, fields contain valid range values.)</p> <ul style="list-style-type: none"> [7:5] – reserved [4] – 1b = ratedMin field supported [3] – 1b = ratedMax field supported [2] – 1b = normalMin field supported [1] – 1b = normalMax field supported [0] – 1b = nominalValue field supported

Type	Description
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>nominalValue</p> <p>This value presents the nominal value for the parameter that is accepted by the effector. The size of this field is given by the rangeFieldFormat field in this PDR. This value is given directly in the specified units without the use of any conversion formula.</p> <p>For example, if the units are millivolts and the nominalValue is 5000, the nominalValue corresponds to 5000 mV, or 5.000 V. It is possible that the nominal value could be some fraction of the given units for the effector (for example, if the units are volts and the nominal value is 2.5 V). For this reason, the nominalValue can be expressed using a real32.</p> <p>The value is defined as the nominal value for what is being set. The nominalValue is not required to match a value that can be returned as a reading by the effector implementation. For example, if the nominal value for a voltage setting effector was 5.00 V, the nominalValue would typically be reported as 5.00 V even though the closest setting the effector implementation may be able to accept is 5.05 V.</p> <p>A common use of the nominalValue is as a source of part of the identifying “name” for an effector. For example, it is common for voltage effectors to be identified by their nominal reading. So, an effector with a nominal reading of +5.00 V would be referred to as a “+5 V effector”, while one with a nominal reading of +3.3 V would be referred to as a “+3.3 V effector”. The definition of nominalValue in the PDR supports this usage. An application that uses or displays this value will typically elect to round the value to some number of significant digits using an algorithm based on the resolution of the effector. For example, if the nominalValue is given as a real32 as 2.50000 V, but the resolution of the effector is 0.05 V, the nominalValue displayed would typically be rounded as 2.50 V.</p> <p>It is possible that a given effector may not be considered as having a nominal setting, in which case this field should be ignored. For example, a numeric effector that sets a count or size of some parameter may not be considered as having a nominal setting depending on its application.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>normalMax</p> <p>The upper limit of the normal operating range for the parameter that is set by the numeric effector. The setting is considered to be operating outside of normal range when this value is exceeded. For example, if a monitored voltage of a component is specified in its data sheet to have a normal maximum operating range of 4.75 to 5.25 V, this value would be set to 5.25 (assuming the units in the PDR are for volts). This value is given directly in the specified units without the use of any conversion formula. This value is used together with normalMin to indicate the normal operating range for the effector.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>normalMin</p> <p>The lower limit of the normal operating range for the parameter that is set by the numeric effector. Effector thresholds are typically set for a value that is lower than normalMin to accommodate the effects of effector accuracy, tolerance, and resolution, in order to prevent false reporting of an “out-of-range” event state. This value is given directly in the specified units without the use of any conversion formula.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>ratedMax</p> <p>The upper limit of the rated operating range for the parameter that is set by the numeric effector. The monitored parameter is considered to be operating outside of rated operating range when this value is exceeded.</p>
uint8 sint8 uint16 sint16 uint32 sint32 real32	<p>ratedMin</p> <p>The lower limit of the rated operating range for the parameter that is set by the numeric effector. The monitored parameter is considered to be operating outside of rated operating range below this value.</p>

2878 **28.12 Numeric Effector Initialization PDR**

2879 The Numeric Effector Initialization PDR reports the values that are used when a PLDM Effector Sensor is
 2880 initialized by a PLDM Initialization Agent. Table 88 describes the format of this PDR.

2881 **Table 88 – Numeric Effector Initialization PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus
uint16	effectorID ID of the effector relative to the given PLDM Terminus ID
enum8	effectorEnable The operational state of the effector after it has been initialized. This state is written to the effector using the SetEffectorEnable command. special value: {0xFF = do not issue a SetEffectorEnable command to set the Effector Operational State }
bitfield8	initConditions Identifies under which conditions the Initialization Agent must initialize or reinitialize this effector [7:5] – reserved [4] – 1b = PLDM terminus returns to online condition [3] – 1b = System warm resets [2] – 1b = System hard resets [1] – 1b = PLDM subsystem power up [0] – 1b = Initialization Agent controller restart/update (initialize/reinitialize this effector whenever the device that holds the Initialization Agent has been restarted or reinitialized)
enum8	effectorDataSize The bit width of reading and threshold values that the effector returns value: { uint8, sint8, uint16, sint16, uint32, sint32 }
uint8 sint8 uint16 sint16 uint32 sint32	effectorData The numeric value written to the effector. The size of this field is determined by the value of the effectorDataSize field.

2882 **28.13 State Effector PDR**

2883 The State Effector PDR is used to provide information about a PLDM Composite State Effector. Table 89
 2884 describes the format of this PDR.

2885 **Table 89 – State Effector PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus
uint16	effectorID ID of the effector relative to the given PLDM Terminus ID
uint16	entityType The Type value for the entity that is associated with this effector. See 9.1. for more information.
uint16	entityInstanceNumber The Instance Number for the entity that is associated with this effector. See 9.1. for more information.
uint16	containerID The containerID for the containing entity that is associated with this effector. See 9.1. for more information.
uint16	effectorSemanticID This field either identifies a PLDM-defined effector semantic or provides an OEMEffectorSemanticHandle value, depending on what range the value falls in. If the effectorSemanticID field is set to a value in the OEM range, this value does not directly identify a particular vendor-defined semantic but instead is interpreted as an OEMEffectorSemanticHandle that can be used to locate an OEM Effector Semantic PDR that identifies the vendor and provides optional name information for the semantic. See DSP0249 for the definition of Effector Semantic ID values and ranges, and 21.3 for more information. special value: {0x0000 = unspecified }
enum8	effectorInit value: { nolnit, // The Initialization Agent does not take any steps to initialize, // enable, or disable this particular effector. useInitPDR, // The effector has an associated State Effector Initialization PDR // that should be used to initialize the effector. enableEffector, // When the Initialization Agent runs, it enables this effector using // a SetStateEffectorEnables command to set the // operationalState. disableEffector. // When the Initialization Agent runs, it disables this effector using // the SetStateEffectorEnables command. }
bool8	effectorDescriptionPDR true = effector has an effectorDescription PDR false = effector does not have an associated effectorDescription PDR

Type	Description
uint8	<p>compositeEffectorCount</p> <p>The number of state effectors in the terminus that are accessed under the effectorID given in this PDR.</p> <p>value: 0x01 to 0x08</p>
var	<p>possibleStates</p> <p>One instance of State Effector Possible States Fields (see Table 90) for each effector in the PLDM State Effector, up to effectorCount.</p>

2886

Table 90 – State Effector Possible States fields format

Type	Description
uint16	<p>stateSetID</p> <p>A numeric value that identifies the PLDM State Set that is used with this effector.</p>
uint8	<p>possibleStatesSize</p> <p>The number of bytes (M) in the possibleStates bitfield.</p> <p>value: 0x01 to 0x20</p> <p>special value : 0x00 can be used to indicate a effector that is unavailable or disabled from use and should be ignored when accessing the parent composite effector with PLDM.</p>
bitfield8 x M	<p>possibleStates [subset of the State Set that is supported]</p> <p>A variable length bitfield that consists of one or more bytes, based on the size of the state set. If stateSetSize is non-zero, possibleStates consists of one or more 8-bit fields where X=0 for the first field, X=1 for the second field (if any), and so on, up to M fields as required by the size of the largest value in the state set.</p> <p>For example, if the largest value in the state set is 7 or less, this will be a one-byte bitfield. If the largest value in the state set is 15 or less, this will be a two-byte bitfield, and so on.</p> <p>The value 0b is also used when no state set value corresponds to the corresponding bit position. For example, if a state set has a maximum value of 5, bits [6] and [7] are unused and shall be set to 0b.</p> <p>[7] – 1b = state that corresponds to value X*8+7 in the state set is supported 0b = state that corresponds to value X*8+7 in the state set is not supported</p> <p>...</p> <p>[2] – 1b = state that corresponds to value X*8+2 in the state set is supported 0b = state that corresponds to value X*8+2 in the state set is not supported</p> <p>[1] – 1b = state that corresponds to value X*8+1 in the state set is supported. 0b = state that corresponds to value X*8+1 in the state set is not supported</p> <p>[0] – 1b = state that corresponds to value X*8+0 in the state set is supported 0b = state that corresponds to value X*8+0 in the state set is not supported</p>

2887

28.14 State Effector Initialization PDR

2888

The State Effector Initialization PDR describes settings that the Initialization Agent uses to initialize a

2889

PLDM Single or Composite State Effector.

2890 The PDR always has eight state values. Dummy values must be used (0x00 is recommended) if the
 2891 implementation does not have an effector that corresponds to a particular offset. Table 91 describes the
 2892 format of the PDR.

2893 **Table 91 – State Effector Initialization PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus
uint16	effectorID ID of the effector relative to the given PLDM terminus
uint16	entityType The Type value for the entity that is associated with this effector. See 9.1 for more information. This field has been deprecated and may be deleted in a future version of this specification. Termini should set this value to zero, and this value should be ignored by readers.
uint16	entityInstanceNumber The Instance Number for the entity that is associated with this effector. See 9.1 for more information. This field has been deprecated and may be deleted in a future version of this specification. Termini should set this value to zero, and this value should be ignored by readers.
uint16	containerID The containerID for the containing entity that is associated with this effector. See 9.1 for more information. This field has been deprecated and may be deleted in a future version of this specification. Termini should set this value to zero, and this value should be ignored by readers.
bitfield8	initConditions Identifies the conditions under which the Initialization Agent must initialize or reinitialize this effector [7:5] – reserved [4] – 1b = PLDM terminus returns to online condition [3] – 1b = System warm resets [2] – 1b = System hard resets [1] – 1b = PLDM subsystem power up [0] – 1b = Initialization Agent controller restart/update (initialize/reinitialize this effector whenever the device that holds the Initialization Agent has been restarted or reinitialized)
enum8	effectorEnable The operational state of the overall composite state sensor after it has been initialized. This state is written to the sensorOperationalState of each sensor that is identified for initialization through the effectorInitMask field of this PDR using the SetStateEffectorEnables command. special value: {0xFF = do not set the effectorOperationalStates}

Type	Description
bitfield8	<p>effectorInitMask</p> <p>Identifies which effecters within the composite state effector require initialization</p> <p>[7] – 1b = state effector at offset 7 requires initialization 0b = state effector at offset 7 does not require initialization</p> <p>[6] – 1b = state effector at offset 6 requires initialization 0b = state effector at offset 6 does not require initialization</p> <p>...</p> <p>[2] – 1b = state effector at offset 2 requires initialization 0b = state effector at offset 2 does not require initialization</p> <p>[1] – 1b = state effector at offset 1 requires initialization 0b = state effector at offset 1 does not require initialization</p> <p>[0] – 1b = state effector at offset 0 requires initialization 0b = state effector at offset 0 does not require initialization</p>
bitfield8	<p>effectorOpStateEventEnableMask</p> <p>Identifies which sensors within the composite state effector should have their operational state event message generation enabled after initialization</p> <p>[7] – 1b = enable event message generator for state sensor at offset 7 0b = disable event message generator for state sensor at offset 7</p> <p>[6] – 1b = enable event message generator for state sensor at offset 6 0b = disable event message generator for state sensor at offset 6</p> <p>...</p> <p>[2] – 1b = enable event message generator for state sensor at offset 2 0b = disable event message generator for state sensor at offset 2</p> <p>[1] – 1b = enable event message generator for state sensor at offset 1 0b = disable event message generator for state sensor at offset 1</p> <p>[0] – 1b = enable event message generator for state sensor at offset 0 0b = disable event message generator for state sensor at offset 0</p>
uint8	<p>stateValue0</p> <p>State value to write to effector offset 0 for initialization special value: Use 0x00 as a placeholder value for effecters that do not require initialization.</p>
uint8	<p>stateValue1</p> <p>State value to write to effector offset 1 for initialization special value: Use 0x00 as a placeholder value for effecters that do not require initialization.</p>
uint8	<p>stateValue2</p> <p>State value to write to effector offset 2 for initialization special value: Use 0x00 as a placeholder value for effecters that do not require initialization.</p>
	<p>...</p>
uint8	<p>stateValue6</p> <p>State value to write to effector offset 6 for initialization special value: Use 0x00 as a placeholder value for effecters that do not require initialization.</p>
uint8	<p>stateValue7</p> <p>State value to write to effector offset 7 for initialization special value: Use 0x00 as a placeholder value for effecters that do not require initialization.</p>

2894 **28.15 Effector Auxiliary Names PDR**

2895 The Effector Auxiliary Names PDR may be used to provide optional information that names an effector.
 2896 This record may be used for a single effector or multiple effectors if the effector is a composite state
 2897 effector.

2898 The nameLanguageTag field can be used to identify the language (such as French, Italian, or English)
 2899 that is associated with the particular effector name. Table 92 describes the format of this PDR.

2900 **Table 92 – Effector Auxiliary Names PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus
uint16	effectorID ID of the effector relative to the given PLDM terminus
uint8	effectorCount [1..M] For each effector x in effectorCount, there can be 1..nameStringCount[x] strings, where each set of strings corresponds to a effector in a composite effector. The record must be populated sequentially starting from 1 regardless of whether an effector requires auxiliary names. Thus, each entry has at least one byte (the nameStringCount). Effectors that have offsets that are greater than effectorCount are treated as if they have no auxiliary names. For example, if a composite effector contains four effectors and only the third effector requires an auxiliary name, the effectorCount can be 3 and the nameStringCount for the first two sets of effector name information is 0.
effector [1] names:	
uint8	nameStringCount Number of following pairs [0..N] of nameLanguageTag + effectorName fields for effector[1].
strASCII	nameLanguageTag[1] This field is absent if nameStringCount = 0. A null-terminated ISO646 ASCII string that holds a language tag, per RFC4646 , that identifies the primary language in which the effectorName was defined (for example, "en" for English, "zh-cmn-Hans" for simplified Mandarin Chinese, and so on). This field may be used to help select which string to use when multiple character encodings for effectorName are provided. special value: null string = 0x0000 = unspecified
strUTF-16BE	effectorName[1] This field is absent if nameStringCount = 0. A null-terminated unicode string for the name of the auxiliary effector special value: null string = 0x0000 = name not provided.
...	...
strASCII	nameLanguageTag[N]
strUTF-16BE	effectorName[N]
effector [2] names:	
...	
effector [M] names:	

2901 **28.16 OEM Effector Semantic PDR**

2902 The OEM Effector Semantic PDR is used to provide information about an OEM effector semantic used
 2903 with one or more PLDM effectors that are represented in the PDRs. The information includes an ID for the
 2904 vendor and a vendor-defined ID number for identifying the effector semantic. The PDR also allows one or
 2905 more descriptive name strings to be provided for the vendor-defined effector semantic. The name strings
 2906 may be provided in different character sets and languages.

2907 The OEMEffectorSemanticHandle value in the PDR is used by other PDRs, such as the PLDM State
 2908 Effector PDR, to point to the particular PLDM OEM Effector Semantic PDR within the PDR Repository.
 2909 OEMStateSetIDHandle values are thus required to be unique for each different PLDM OEM State Set
 2910 PDR within a given PDR Repository.

2911 The OEMSemanticID field enables the vendor that defined the semantic to assign an ID value to its
 2912 semantic. The OEMSemanticID field is thus defined relative to the given vendor ID.

2913 The OEM Effector Semantic PDR also contains a PLDMTerminusHandle value. The
 2914 PLDMTerminusHandle is used to provide a record of the terminus from which the PDR was imported. It is
 2915 expected that most vendors will define their OEMSemanticID values in a global manner in which the ID
 2916 has the same meaning regardless of the PLDMTerminusHandle value.

2917 Table 93 describes the format of this PDR.

2918 **Table 93 – OEM Effector Semantic PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle This value is used to identify the terminus that originated this PDR.
uint8	OEMEffectorSemanticHandle An opaque number that is used to identify different OEM effector semantics that are defined by the given vendor on the given terminus. The value is used in PDRs such as the PLDM State Effector PDR to indicate that a vendor-defined effector semantic is being used and to locate the PLDM OEM Effector Semantic PDRs (if any) that provide the vendor-defined ID number and optional descriptive names for the effector semantic.
uint32	vendorIANA The IANA Enterprise Number for the vendor that is defining the OEM Sensor Unit
uint8	OEMEffectorSemanticID A value that can be used as a search field for the FindPDR command. This number is assigned by the vendor and provides a numeric ID for the vendor-defined effector semantic. Thus, the vendor can use this value to provide a constant ID that always identifies a particular Unit definition from that vendor.
uint8	stringCount The number 1..N of languageTag and name field pairs that follow this field. { 0 = no name information provided }
strASCII	languageTag[1] A null-terminated ISO646 ASCII string that holds a language tag, per RFC4646 , that identifies the primary language in which the unitName was defined (for example, "en" for English, "zh-cmn-Hans" for simplified Mandarin Chinese, and so on). This field may be used to help select which string to use when multiple character encodings for the unitName are provided. special value: null string = unspecified
strUTF-16BE	name[1] A null-terminated unicode string that contains the name of the OEM Sensor Unit
...	...
strASCII	languageTag[N]

Type	Description
strUTF-16BE	name[N]

2919 **28.17 Entity Association PDR**

2920 The Entity Association PDR is used to form associations between entities, such as physical and logical
 2921 entities. See clause 10 for more information. Table 94 describes the format of this PDR.

2922 **Table 94 – Entity Association PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	containerID value: 0x0001 to 0xFFFF = An opaque number that identifies a particular container entity in the hierarchy of containment. See 11.1 for more information. special value: 0x0000 = "SYSTEM". This value is used to identify the topmost containing entity in PLDM Entity Association containment hierarchies.
enum8	associationType value: { physicalToPhysicalContainment, logicalContainment }
<i>Container Entity Identification Information</i>	
uint16	containerEntityType
uint16	containerEntityInstanceNumber A top-level PDR shall use containerEntityInstanceNumber 1. Any sensor which relates to this level shall use the containerEntityType and containerEntityInstanceNumber to reference the top level. This method should only be used on the top-level entity association PDR.
uint16	containerEntityContainerID
<i>Contained Entity Identification Information</i>	
uint8	containedEntityCount The number of contained entities (1 to N) listed in this particular PDR. This may not be the total number of contained entities because multiple containment association PDRs may exist for the same container entity. See 11.3 for more information.
uint16	containedEntityType[1]
uint16	containedEntityInstanceNumber[1]
uint16	containedEntityContainerID[1]
	...
uint16	containedEntityType[N]
uint16	containedEntityInstanceNumber[N]
uint16	containedEntityContainerID[N]

2923 **28.18 Entity Auxiliary Names PDR**

2924 The Entity Auxiliary Names PDR may be used to provide optional information that names a particular
 2925 instance of an entity. The PDR can also be used to name a particular range of instances of an entity,
 2926 provided that the instances share the same containerID.

2927 The nameLanguageTag field can be used to identify the language (such as French, Italian, or English)
 2928 that is associated with the particular entity name. Table 95 describes the format of this PDR.

2929 **Table 95 – Entity Auxiliary Names PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	entityType
uint16	entityInstanceNumber
uint16	entityContainerID
uint8	sharedNameCount This number is added to the EntityInstanceNumber to identify how many additional EntityInstanceNumber values share this auxiliary name PDR, where EntityInstanceNumber identifies the starting value for the range. For example, if the EntityInstanceNumber is 100 and the sharedNameCount is 2, this PDR applies to EntityInstanceNumbers 100, 101, and 102. If the sharedNameCount is 0, this PDR applies only to the given EntityInstanceNumber.
Entity auxiliary names:	
uint8	nameStringCount Number of following pairs [0..N] of nameLanguageTag + entityAuxName fields for entityAuxName[1].
strASCII	nameLanguageTag [1] This field is absent if nameStringCount = 0. A null-terminated ISO646 ASCII string that holds a language tag, per RFC4646 , that identifies the primary language in which the entityAuxName was defined (for example, "en" for English, "zh-cmn-Hans" for simplified Mandarin Chinese, and so on). This field may be used to help select which string to use when multiple character encodings for the entityAuxName are provided. special value: null string = 0x0000 = unspecified
strUTF-16BE	entityAuxName [1] This field is absent if nameStringCount = 0. A null-terminated unicode string for the auxiliary name of the entity. special value: null string = 0x0000 = name not provided
...	...
strASCII	nameLanguageTag [N]
strUTF-16BE	entityAuxName [N]

2930 **28.19 OEM EntityID PDR**

2931 The OEM EntityID PDR can be used to provide a vendor-specific EntityID definition when no PLDM
 2932 predefined EntityID corresponds to the type of entity that the vendor wants to represent.

2933 When the entityType value is in the OEM range of values, the EntityID portion of the entityType field is
 2934 OEM-defined. The EntityID value is then used as an OEMEntityIDHandle to locate the corresponding
 2935 OEM EntityID PDR.

2936 OEM Entity Type PDRs need to be able to be exported by a terminus, such as a terminus on a hot-plug
 2937 card. The numbers in a given vendor's Device PDRs must be picked a priori by the vendor. Thus,
 2938 duplications may exist among the OEM EntityID values that different vendors choose. The Discovery
 2939 Agent function is responsible for adjusting the OEM Entity Type values to resolve any conflicts that may
 2940 occur when it integrates PDRs into the Primary PDR Repository. Users of OEM EntityID values must be
 2941 aware that these values may differ between different PDR Repositories. That is, an OEM EntityID for
 2942 "widget" from vendor "ABC" will not always have the same Entity ID value across PDRs.

2943 To facilitate the identification of particular OEM EntityIDs from a given vendor, each PDR includes a
 2944 vendor-specific ID value that does not get altered by the Discovery Agent function. When used in
 2945 conjunction with the vendor's ID, this provides a value that can always be used to identify the particular
 2946 vendor-defined EntityID definition.

2947 Table 96 describes the format of this PDR.

2948 **Table 96 – OEM EntityID PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle This value is used to identify the terminus that originated this PDR.
uint16	OEMEntityIDHandle [15] – 0b = reserved [14:0] – OEM entityID handle value. The value that is used in entity associations and other PDRs to identify the entity defined by this PDR. This value may be changed if the PDR is migrated and integrated into a Primary PDR Repository.
uint32	vendorIANA The IANA Enterprise Number for the vendor that is defining the OEM PDR vendor-specific data
uint16	vendorEntityID This value can be used as a search field for the FindPDR command. This number is assigned by the vendor and provides a numeric ID for the vendor-defined entity. This field is intended to provide a consistent and constant ID that can be relied on to identify the vendor-defined entity even if the name strings need to be changed or updated. [15] – 0b = reserved [14:0] – vendorEntityID value
uint8	stringCount The number 1..N of entityIDLanguageTag and entityIDName field pairs that follow this field.

Type	Description
strASCII	entityIDLanguageTag[1] A null-terminated ISO646 ASCII string that holds a language tag, per RFC4646 , that identifies the primary language in which the EntityID name was defined (for example, "en" for English, "zh-cmn-Hans" for simplified Mandarin Chinese, and so on). This field may be used to help select which string to use when multiple character encodings for the entityIDName are provided. special value: null string = unspecified
strUTF-16BE	entityIDName[1] A null-terminated unicode string that contains the name of the EntityID name
...	...
strASCII	entityIDLanguageTag[N]
strUTF-16BE	entityIDName[N]

2949 **28.20 Interrupt Association PDR**

2950 The Interrupt Association PDR is used to form associations between interrupt source entities and interrupt
2951 target entities. See 11.10 for more information. Table 97 describes the format of this PDR.

2952 **Table 97 - Interrupt Association PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle This value is used to identify the terminus that provides access to the sensor that is monitoring the interrupt that is related to this association.
uint16	sensorID The ID of the sensor that monitors this interrupt at a source or target
enum8	sourceOrTargetSensor Identifies whether the sensor is monitoring the interrupt at the source or the target. The association record for a sensor that monitors an interrupt source is required to identify only a single target entity and a single source entity. value: { targetSensor, sourceSensor }
<i>Target Entity Identification Information</i>	
uint16	interruptTargetEntityType
uint16	interruptTargetEntityInstanceNumber
uint16	interruptTargetEntityContainerID
<i>Source Entity Identification Information</i>	
uint8	interruptSourceEntityCount The number of interruptSource entities (1 to N) listed in this particular PDR. This number may not be the total number of interruptSource entities associated with a particular interrupt target entity because multiple interrupt association PDRs may exist for the same target entity. See 11.3 and 11.10 for more information.
uint32	interruptSourcePLDMTerminusHandle[1]

Type	Description
uint16	interruptSourceEntityType[1]
uint16	interruptSourceEntityInstanceNumber[1]
uint16	interruptSourceEntityContainerID[1]
uint16	interruptSourceSensorID[1]
	...
uint32	interruptSourcePLDMTerminusHandle[N]
uint16	interruptSourceEntityType[N]
uint16	interruptSourceEntityInstanceNumber[N]
uint16	interruptSourceEntityContainerID[N]
uint16	interruptSourceSensorID[N]

2953 **28.21 Event Log PDR**

2954 The Event Log PDR is used to describe characteristics of the PLDM Event Log (if implemented). The
 2955 specification defines the existence of only a single, central PLDM Event Log function. Therefore, only one
 2956 occurrence of a PLDM Event Log PDR shall exist in a Primary PDR Repository.

2957 Table 98 describes the format of this PDR.

2958 **Table 98 – Event Log PDR format**

Type	Description
–	commonHeader See 28.1.
uint32	logSize The size in bytes of the log storage area that is used for storing log entries. This number is exclusive of any fixed overhead for maintaining the overall log, but may include per entry overhead. special value: { 0x0000_0000 = unspecified. 0xFFFF_FFFE = reserved for future definition 0xFFFF_FFFF = log size is greater than or equal to 4 GB-1 bytes }
bitfield8	supportedLogClearingPolicies See 13.4 for a description of the log clearing policies. [7:3] – reserved [2] – 1b = clearOnAge supported [1] – 1b = FIFO supported [0] – 1b = fillAndStop supported

Type	Description
uint8	<p>entryIDTimeout</p> <p>The minimum interval, in seconds, that the entryID used in the middle of a partial transfer remains valid after it was delivered in the response for a GetPLDMEventLogEntry command that returns partial data. This corresponds to the entryID value returned in any GetPLDMEventLogEntry responses where the splitEntry field in the response is firstFragment or middleFragment.</p> <p>special values: { 0x00 = no timeout, 0x01 = default minimum timeout is the same as the PDR Handle Timeout, MC1, (see clause 28.25), 0xFF = timeout >254 seconds. Any timeout values that are less than the specified default minimum timeout are illegal. }</p>
uint8	<p>perEntryOverhead</p> <p>The number of bytes of storage overhead per entry if that overhead is counted as using space from the log area specified by logSize. For example, if this value is 2 and an N-byte entry was added to the log, the amount of logSize consumed would be N+2 bytes.</p> <p>An implementation may elect to hide some or all of the impact of per-entry overhead on the available log space. For example, the implementation may have an internal overhead of 2 bytes but keep that overhead in a separate data structure that does not affect the amount of space consumed from the log. In this case, adding an N-byte entry to the log would be counted as consuming only N-bytes of log space, not N+2 bytes.</p> <p>special value: 0xFF = unspecified</p>
uint8	<p>allocationGranularity</p> <p>The byte multiple or increment by which storage space is allocated to entries. This value typically corresponds to some byte, word, or block boundary related to the physical medium used for storing entries. For example, if this value is 16 and a 24-byte entry were added, the result would be that the entry would consume 32-bytes of storage space.</p> <p>special value: 0xFF = unspecified</p>
uint8	<p>percentUsedResolution</p> <p>Indicates the resolution of the storagePercentUsed value from the GetPLDMEventLogInfo command</p> <p>value: 1 to 100; all other values = reserved</p> <p>A percentUsedResolution value of 0x01 indicates that the storagePercentUsed value is given with a resolution of 1 count (1%), which means a storagePercentUsed value of 0x00 indicates that the log is from 0 to <1% full, a storagePercentUsed value of 0x01 indicates that the log is 1% to <2% full, and so on.</p> <p>A percentUsedResolution value of 0x05 indicates that the storagePercentUsed value is given with a resolution of 5 count (5%), which means a storagePercentUsed value of 0x00 indicates that the log is from 0 to <5% full, a storagePercentUsed value of 0x01 indicates that the log is 5% to <10% full, and so on.</p>

2959 **28.22 FRU Record Set PDR**

2960 The FRU Record Set PDR is used to describe characteristics of the PLDM FRU Record Set Data defined
 2961 in [DSP0257](#). The information can be used to locate a Terminus that holds FRU Record Set Data in order
 2962 to access that data using the commands specified in [DSP0257](#). The PDR also identifies the particular
 2963 Entity that is associated with the FRU information.

2964 Table 99 describes the format of this PDR.

2965

2966

Table 99 – FRU Record Set PDR format

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle The terminus that originated or maintains this PDR. .
uint16	FRURecordSetIdentifier A unique number per terminus that is used to identify the Record Set for the FRU Data for the associated entity. The Record Set value is used for accessing FRU Data using the commands specified in DSP0257 .
uint16	entityType The Type value for the entity that is associated with this FRU data.
uint16	entityInstanceNumber The Instance Number for the entity that is associated with this FRU data.
uint16	containerID The containerID for the containing entity that is associated with this FRU data.

2967 **28.23 OEM Device PDR**

2968 The OEM Device PDR can be used to provide OEM (vendor-specific) information. The OEM-specific data
 2969 portion in an OEM Device PDR is limited to a maximum size of 64 KB. Higher-level system specifications
 2970 may place additional limits on the size and number of OEM Device PDRs that may be supported in a
 2971 given PLDM subsystem implementation. An OEM Device PDR must have at least one byte of
 2972 VendorSpecificData.

2973 This type of PDR shall be copied by the Discovery Agent into the Primary PDR Repository dependent on
 2974 the setting of the copyPDR field. The PDR may also be preconfigured into the Primary PDR Repository.
 2975 That is, this PDR is not restricted to being only used or migrated from repositories that are separate from
 2976 the Primary PDR Repository.

2977 The OEM PDR is a slightly smaller version of the OEM Device PDR that can be used in situations where
 2978 it is not necessary or desired to associate the PDR to a particular terminus or have the information copied
 2979 from a Device PDR Repository into the Primary PDR Repository.

2980 Table 100 describes the format of this PDR.

2981 **28.23.1 Copy Behavior**

2982 If the copyPDR parameter is set to copyToPrimaryRepository, the Discovery Agent shall overwrite any
 2983 pre-existing PDRs for the terminus that have the same vendorIANA and VendorHandle values.

2984 **28.23.2 Removal Behavior**

2985 The OEM Device PDR is allowed to be removed from the Primary PDR Repository if the Discovery Agent
 2986 detects that the terminus that is associated with the PDR has been removed or is no longer available.

2987

Table 100 – OEM Device PDR format

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle The PLDMTerminusHandle for the terminus from which this record was obtained. special value: 0x0000 may be used to indicate 'unspecified' when this record is in a device's PDR Repository. The Discovery Agent typically assigns a different value to this field when merging the record into the Primary PDR Repository.
enum8	copyPDR value: { doNotCopy, copyToPrimaryRepository }
uint32	vendorIANA The IANA Enterprise Number for the vendor that is defining the OEM PDR vendor -specific data special value: 0 = unspecified
uint16	OEMRecordID This value can be used as a search field for the FindPDR command. This value must be unique among all OEM Device PDRs for a given terminus that share the same vendorIANA value. Any other semantics associated with this value are vendor-specific and defined by the vendor or group that is identified by vendorIANA.
uint16	dataLength The number of following vendorSpecificData bytes starting from 0. 0 = 1 byte, 1 = 2 bytes, and so on
byte	vendorSpecificData[0]
...	...
byte	vendorSpecificData[N]

2988 **28.24 OEM PDR**

2989 The OEM PDR can be used to provide OEM (vendor-specific) information. The OEM-specific data portion
 2990 in an OEM PDR is limited to a maximum size of 64 KB. Higher-level system specifications may place
 2991 additional limits on the size and number of OEM PDRs that may be supported in a given PLDM
 2992 subsystem implementation. An OEM PDR must have at least one byte of vendorSpecificData. The OEM
 2993 Device PDR is an extended version of the OEM PDR that is used when it is necessary to associate the
 2994 PDR to a particular terminus or to have the information copied from a Device PDR Repository into the
 2995 Primary PDR Repository.

2996 Table 101 describes the format of this PDR.

2997

Table 101 – OEM PDR format

Type	Description
–	commonHeader See 28.1.

Type	Description
uint32	vendorIANA The IANA Enterprise Number for the vendor that is defining the OEM PDR vendor-specific data special value: 0 = unspecified
uint16	OEMRecordID This value can be used as a search field for the FindPDR command. This value must be unique among all OEM PDRs within the PDR Repository that share the same vendorIANA value. Any other semantics associated with this value are vendor-specific and defined by the vendor or group that is identified by vendorIANA.
uint16	dataLength The number of following vendor-specific data bytes starting from 0 0 = 1 byte, 1 = 2 bytes, and so on.
byte	vendorSpecificData[1]
...	...
byte	vendorSpecificData[N]

2998 **28.25 Compact Numeric Sensor PDR**

2999 The Compact Numeric Sensor PDR is designed for Management Controller (MC) monitoring of a
 3000 sophisticated PLDM terminus (device) where data conversion is not required. This sensor always reports
 3001 normalized integer values. Temperature and counting sensors are examples of sensor types that may be
 3002 defined by this PDR sensor type. Any mapping to an external management protocol is defined outside of
 3003 this specification.

3004 The commands, which specify a “raw value” such as SetSensorThresholds, GetSensorThresholds and
 3005 GetSensorReading, shall use the sensor’s (integer) value.

3006 This sensor is for simple numeric sensor reporting. For complex designs, the standard Numeric Sensor
 3007 PDR is retained and supported.

3008 **Table 102 – Compact Numeric Sensor PDR format**

Type	Description
–	commonHeader See 28.1.
uint16	PLDMTerminusHandle A handle that identifies PDRs that belong to a particular PLDM terminus.
uint16	sensorID ID of the sensor relative to the given PLDM Terminus ID.
uint16	entityType The Type value for the entity that is associated with this sensor. See 9.1 for more information.
uint16	entityInstanceNumber The Instance Number for the entity that is associated with this sensor. See 9.1 for more information.
uint16	containerID The containerID for the containing entity that is associated with this sensor. See 9.1 for more information.

Type	Description
uint8	<p>sensorNameStringByteLength</p> <p>If this is greater than zero, then the “sensorNameString” is present at the end of this PDR. This field is a vendor supplied sensor name. This is the an explicit name for display. The recommended maximum length is 96 bytes.</p>
enum8	<p>baseUnit</p> <p>The base unit of the reading returned by this sensor. See 27.4 for more information.</p> <p>value: { see Table 74 }</p>
sint8	<p>unitModifier</p> <p>A power-of-10 multiplier for the baseUnit. See 27.4 for more information.</p>
enum8	<p>occurrenceRate</p> <ul style="list-style-type: none"> 0 : No Occurrence Rate 1 : Per Microsecond 2 : Per Millisecond 3 : Per Second 4 : Per Minute 5 : Per Hour 6 : Per Day
bitfield8	<p>rangeFieldSupport</p> <p>Indicates which of the fields that identify the operating ranges of the parameter monitored by the sensor are supported. (This bitfield indicates whether the following threshold fields contain valid range values).</p> <p>[6:7] – reserved</p> <p>[5] – 1b = fatalLow field supported</p> <p>[4] – 1b = fatalHigh field supported</p> <p>[3] – 1b = criticalLow field supported</p> <p>[2] – 1b = criticalHigh field supported</p> <p>[1] – 1b = warningLow field supported</p> <p>[0] – 1b = warningHigh field supported</p>
sint32	<p>warningHigh</p> <p>A warning condition that occurs when the monitored value is <i>greater than</i> the value reported by warningHigh. In many implementations, this value may be the same value as normalMax. Sensor thresholds that may be derived from this value are typically set for a value that is higher than warningHigh to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.</p>
sint32	<p>warningLow</p> <p>A warning condition that occurs when the monitored value is <i>less than or equal to</i> the value reported by warningLow. In many implementations, this value may be the same value as normalMin. Sensor thresholds that may be derived from this value are typically set for a value that is lower than warningLow to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.</p>

Type	Description
sint32	criticalHigh A critical condition that occurs when the monitored value is <i>greater than or equal to</i> the value reported by criticalHigh. In some implementations, this value may be the same value as normalMax. Sensor thresholds that may be derived from this value are typically set for a value that is higher than criticalHigh to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.
sint32	criticalLow A critical condition that occurs when the monitored value is <i>less than</i> the value reported by criticalLow. In some implementations, this value may be the same value as normalMin. Sensor thresholds that may be derived from this value are typically set for a value that is lower than criticalLow to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.
sint32	fatalHigh A fatal condition that occurs when the monitored value is <i>greater than</i> the value reported by fatalHigh. In many implementations, this value may be the same value as normalMax. Sensor thresholds that may be derived from this value are typically set for a value that is higher than fatalHigh to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.
sint32	fatalLow A fatal condition that occurs when the monitored value is <i>less than</i> the value reported by fatalLow. In many implementations, this value may be the same value as normalMin. Sensor thresholds that may be derived from this value are typically set for a value that is lower than fatalLow to accommodate the effects of sensor accuracy, tolerance, and resolution, in order to prevent false reporting of an out-of-range condition. This value is given directly in the specified units without the use of any conversion formula.
strUTF-8	sensorNameString This is the vendor defined name for this sensor. This field is expected to be use for display and not an explicit identifier. This field is NOT present if the sensorNameStringByteLength value is equal to zero.

3009

3010 **28.26 Redfish Resource PDR**

3011 The Redfish Resource PDR provides the Redfish Schema information for every Redfish resource
 3012 managed by a data provider. The usage of this PDR is defined in [DSP0218](#), *Platform Level Data Model*
 3013 *for Redfish Device Enablement*.

3014 **Table 103 – Redfish Resource PDR format**

Type	Description
–	CommonHeader See [28.1].
uint32	ResourceID The primary resourceID for this collection of data. All ResourceIDs (including those in the AdditionalResourceID field below) across all Redfish Resource PDRs presented by an RDE Device shall be unique to that device.

Type	Description
bitfield8	<p>ResourceFlags</p> <p>Flags associated with this Resource:</p> <p>[7:3] - reserved for future use</p> <p>[2] - is_collection; if 1b, this resource is a Redfish collection that contains zero or more resources sharing a common schema</p> <p>[1] - is_contained_in_collection; if 1b, the resource in which this resource is contained is a collection</p> <p>[0] - is_device_root; if 1b, this resource is a root of the RDE Device's logical containment hierarchy and shall have ContainingResourceID below set to EXTERNAL</p>
uint32	<p>ContainingResourceID</p> <p>value: 0x0000 0001 to 0xFFFF FFFE = An opaque number that references a Redfish Resource PDR in the hierarchy of containment. See DSP0218 for more information.</p> <p>special value: 0x0000 0000 = "EXTERNAL". This value is used to identify the logical root of a device component's management topology.</p> <p>special value: 0xFFFF FFFF is reserved for special use within DSP0218.</p>
uint16	<p>ProposedContainingResourceLengthBytes</p> <p>Length in bytes of the proposed parent resource that the resource this PDR represents should be subordinate to. Shall be 1 if ContainingResourceID is not EXTERNAL</p>
strUTF-8	<p>ProposedContainingResourceName</p> <p>Name of the schema for the proposed parent resource to which this PDR's primary resource (and any additional resources) should be subordinate. Shall be a null byte if ContainingResourceID is not EXTERNAL. The MC may accept or reject this placement recommendation at its discretion. The format and usage of this field is defined in DSP0218, <i>Platform Level Data Model for Redfish Device Enablement</i>.</p> <p>The name specified shall be the fully qualified Odata name, in the format <i>Namespace.EntityType</i>. For example, a storage controller might specify StorageCollection.StorageCollection as its proposed containing resource name.</p>
uint16	<p>SubURLengthBytes</p> <p>Length in bytes of the SubURI path fragment (including the null terminator) for the primary resource</p>
strUTF-8	<p>SubURI</p> <p>Null-terminated SubURI path fragment corresponding to the primary resource's portion of the canonical OpenAPI pathname for this resource. Shall neither begin nor end with a slash ('/') character. Shall be a null byte if ContainingResourceID is EXTERNAL.</p> <p>To define the contents for this field, let:</p> <ul style="list-style-type: none"> • P_P (parent path) be the standardized OpenAPI path for the Redfish resource containing this resource • P_R (resource path) be the standardized OpenAPI path for this resource <p>The subURI for this field shall be the difference (P_R – P_P). In most cases it will consist of a single path segment, but may consist of several slash-separated segments.</p> <p>For example, the OpenAPI path for a NetworkPortCollection (P_R) is /redfish/v1/Chassis/{ChassisID}/NetworkAdapters/{NetworkAdapterID}/NetworkPorts.</p> <p>P_P is /redfish/v1/Chassis/{ChassisID}/NetworkAdapters/{NetworkAdapterID}.</p> <p>The SubURI for this case would be "NetworkPorts".</p> <p>For further details on the usage of this field, please refer to DSP0218, <i>Platform Level Data Model for Redfish Device Enablement</i>.</p>

Type	Description
uint16	AdditionalResourceIDCount Number N_A of additional resourceIDs, each of which represents a separate instance of a Redfish resource that shares all the same schema data with the primary resourceID
uint32	AdditionalResourceID [0] The resourceID for another resource instance that shares all the same schema data detailed in this PDR with the primary resource instance. All ResourceIDs across all Redfish Resource PDRs presented by an RDE Device shall be unique to that device.
uint16	AdditionalResourceSubURLengthBytes [0] Length in bytes of the SubURI path fragment (including the null terminator) for this additional resource
strUTF-8	AdditionalResourceSubURI [0] Null-terminated SubURI path fragment corresponding to this resource's portion of the canonical OpenAPI pathname for this additional resource. Shall neither begin nor end with a slash ('/') character. Shall be a null byte if ContainingResourceID is EXTERNAL. This field shall be formatted according to the rules defined above for the SubURI field.
...	...
uint32	AdditionalResourceID [N_A-1] The resourceID for another resource instance that shares all the same schema data detailed in this PDR with the primary resource instance. All ResourceIDs across all Redfish Resource PDRs presented by an RDE Device shall be unique to that device.
uint16	AdditionalResourceSubURLengthBytes [$N_A - 1$] Length in bytes of the SubURI path fragment (including the null terminator) for this additional resource
strUTF-8	AdditionalResourceSubURI [$N_A - 1$] Null-terminated SubURI path fragment corresponding to this resource's portion of the canonical OpenAPI pathname for this additional resource. Shall neither begin nor end with a slash ('/') character. Shall be a null byte if ContainingResourceID is EXTERNAL. This field shall be formatted according to the rules defined above for the SubURI field.
ver32	MajorSchemaVersion In standard PLDM version format; 0xFFFFFFFF for an unversioned schema
uint16	MajorSchemaDictionaryLengthBytes Length of dictionary data for the major schema
uint32	MajorSchemaDictionarySignature 32-bit CRC for the major schema dictionary, including all OEM extensions. 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 signature computation. The CRC computation involves processing a byte at a time with the least significant bit first.
uint8	MajorSchemaNameLength Length of the name of the major schema, including null terminator
strUTF-8	MajorSchemaName Null-terminated UTF-8 string containing the name of the major schema

Type	Description
uint16	OEMCount Number N_O of OEMs associated with this resource in the device
uint16	OEMNameLengthBytes [0] Length in bytes of OEMName [0], below, including the null terminator
strUTF-8	OEMName [0] Null-terminated UTF-8 string containing the name of the first OEM
...	...
uint16	OEMNameLengthBytes [$N_O - 1$] Length in bytes of OEMName [0], below, including the null terminator
strUTF-8	OEMName [$N_O - 1$] Null-terminated UTF-8 string containing the name of the last OEM

3015 **28.27 Redfish Entity Association PDR**

3016 The Redfish Entity Association PDR provides the topology (or hierarchy) of Redfish (data) resources. The
 3017 usage of this PDR is defined in [DSP0218](#), *Platform Level Data Model for Redfish Device Enablement*.

3018 **Table 104 – Redfish Entity Association PDF format**

Type	Description
–	CommonHeader See 28.1.
<i>Container Entity Identification Information</i>	
uint32	ContainingResourceID value: 0x0000 0001 to 0xFFFFFFFF = An opaque number that references a Redfish Resource PDR in the hierarchy of containment. See DSP0218 for more information. special value: 0x0000 0000 = “EXTERNAL”. This value is used to identify the topmost containing entity for a device component in PLDM Entity Association containment hierarchies. special value: 0xFFFF FFFF is reserved for special use within DSP0218 .
uint16	ProposedContainingResourceLengthBytes Length in bytes of the proposed parent resource that the resource this PDR represents should be subordinate to. Shall be 1 if ContainingResourceID is not EXTERNAL
utf8string	ProposedContainingResourceName Name of the proposed parent resource that the resource this PDR represents should be subordinate to. Shall be null (“”) if ContainingResourceID is not EXTERNAL. The MC may accept or reject this placement recommendation at its discretion.
<i>Contained Entity Identification Information</i>	
uint8	ContainedEntityCount The number of contained entities N_C listed in this particular PDR. This may not be the total number of contained entities because multiple containment association PDRs may exist for the same container entity. See 11.3 for more information.
uint32	ContainedEntityResourceID [0]

Type	Description
	...
uint32	ContainedEntityResourceID [$N_C - 1$]

3019 **28.28 Redfish Action PDR**

3020 The Redfish Action PDR provides the details of the “Actions” a resource can execute. The “Actions” are
 3021 described in standard Redfish resource schema definition. The usage of this PDR is defined in DSP0218
 3022 Platform Level Data Model for Redfish Device Enablement.

3023 **Table 105 – Redfish Action PDR format**

Type	Description
–	CommonHeader See 28.1.
uint8	ActionPDRIndex Zero-based index for Action PDRs linked to a single Redfish Resource PDR; this established an ordering on the Actions in the event that they are split across multiple Redfish Action PDRs.
<i>Host Resource Information</i>	
uint16	RelatedResourceCount The number N_R of Resources the Actions in this PDR are being linked to. If listing the full number of related resources would cause this PDR to exceed the maximum supported PDR size, the PDR may be split into multiple copies, each listing a subset of the related resources. Splitting related resources should be employed in preference to splitting actions for the same resource.
uint32	RelatedResourceID [0] value: 0x0000 0001 to 0xFFFF FFFE = An opaque number that identifies the Redfish Resource PDR in which the Action is defined. Values 0x0000 0000 and 0xFFFF FFFF are reserved.
...	...
uint32	RelatedResourceID [$N_R - 1$] value: 0x0000 0001 to 0xFFFF FFFE = An opaque number that identifies the Redfish Resource PDR in which the Action is defined. Values 0x0000 0000 and 0xFFFF FFFF are reserved.
<i>Action Information</i>	
uint8	ActionCount The number of Redfish Actions N_A associated with the host Redfish Resource PDR. If listing all of the actions for a resource would cause this PDR to exceed the maximum supported PDR size, the PDR may be split into multiple copies, each listing a subset of the supported actions. Splitting actions in this fashion should only be done if the actions themselves cannot fit within a single PDR; PDRs should be preferentially split by resource ahead of action.
uint8	ActionNameLengthBytes [0] Including null terminator
utf8string	ActionName [0] The name of action, null-terminated

Type	Description
uint8	ActionPathLengthBytes [0] The length in bytes of the null-terminated string detailing the path to the root of the Action within the resource's major dictionary.
utf8string	ActionPath [0] Null-terminated string detailing the path to the root of the Action within the resource's major dictionary.
	...
uint8	ActionNameLengthBytes [N_A - 1] Including null terminator
utf8string	ActionName [N_A - 1] The name of action, null-terminated
uint8	ActionPathLengthBytes [N_A - 1] The length in bytes of the null-terminated string detailing the path to the root of the Action within the resource's major dictionary.
utf8string	ActionPath [N_A - 1] Null-terminated string detailing the path to the root of the Action within the resource's major dictionary.

3024 **29 Timing**

3025 Table 106 defines timing values that are specific to this document.

3026 **Table 106 – Monitoring and control timing specifications**

Timing specification	Symbol	Min	Max	Description
PDR record handle retention	MC1	30 sec	–	See 26.2.8.

3027 **30 PLDM Command numbers**

3028 Table 107 defines the PLDM command numbers used in the requests and responses for the PLDM
3029 monitoring and control commands defined in this specification.

3030 **Table 107 – Command numbers**

#	Command	Reference
Terminus commands		
0x01	SetTID (see DSP0240)	See 16.1.
0x02	GetTID (see DSP0240)	See 16.2
0x03	GetTerminusUID	See 16.3.
0x04	SetEventReceiver	See 16.4.
0x05	GetEventReceiver	See 16.5.
0x0A	PlatformEventMessage	See 16.6.
0x0B	PollForPlatformEventMessage	See 16.7
0x0C	EventMessageSupported	See 16.8

#	Command	Reference
0x0D	EventMessageBufferSize	See 16.9
Numeric Sensor commands		
0x10	SetNumericSensorEnable	See 18.1.
0x11	GetSensorReading	See 18.2.
0x12	GetSensorThresholds	See 18.3.
0x13	SetSensorThresholds	See 18.4.
0x14	RestoreSensorThresholds	See 18.5.
0x15	GetSensorHysteresis	See 18.6.
0x16	SetSensorHysteresis	See 18.7.
0x17	InitNumericSensor	See 18.8.
State Sensor commands		
0x20	SetStateSensorEnables	See 20.1.
0x21	GetStateSensorReadings	See 20.2.
0x22	InitStateSensor	See 20.3.
PLDM Effector commands		
0x30	SetNumericEffectorEnable	See 22.1.
0x31	SetNumericEffectorValue	See 22.2.
0x32	GetNumericEffectorValue	See 22.3.
0x38	SetStateEffectorEnables	See 22.4.
0x39	SetStateEffectorStates	See 22.5.
0x3A	GetStateEffectorStates	See 22.6.
PLDM Event Log commands		
0x40	GetPLDMEventLogInfo	See 23.1.
0x41	EnablePLDMEventLogging	See 23.2.
0x42	ClearPLDMEventLog	See 23.3.
0x43	GetPLDMEventLogTimestamp	See 23.4.
0x44	SetPLDMEventLogTimestamp	See 23.5.
0x45	ReadPLDMEventLog	See 23.6.
0x46	GetPLDMEventLogPolicyInfo	See 23.7.
0x47	SetPLDMEventLogPolicy	See 23.8.
0x48	FindPLDMEventLogEntry	See 23.9
PDR Repository commands		
0x50	GetPDRRepositoryInfo	See 26.1.
0x51	GetPDR	See 26.2.
0x52	FindPDR	See 26.3.
0x58	RunInitAgent	See 26.4.
0x53	GetPDRRepositorySignature	See 26.5

ANNEX A
(informative)

Change log

3032
3033
3034
3035
3036

Version	Date	Description
1.0.0	2009-03-16	
1.0.1	2010-01-13	Update to correct address issues from TC ballot
1.1.0	2011-11-08	DMTF Standard. Added FRU Record Set PDR and description of FRU Record Set to Entity Association relationship. A 'rel' field that describes the relationship between the base unit and aux unit was added to the Numeric Sensor PDR format. This update also included edits for consistency, typos, and clarifications per Mantis entries, including: References to "effectorDescriptionPDR" and "sensorDescription PDR" in v1.0.x were changed to refer to the EffectorAuxiliaryNames and SensorAuxiliaryNames PDRs, respectively. The enumeration values of effectorOperationalState in Tables 37 and 43 were made consistent. Similarly, the enumeration values for sensorOperationalState in Table 19 & Table 30 were also made consistent. In Table 77, the type of effectorInIt was incorrectly specified as bool8 instead of enum8. In table 19, sensorEventMessageEnable type was specified as bool8 instead of enum8.
1.1.1	2016-12-20	Corrected the data type length of the "sensorID" and corresponding "effectorID" field from "uint8" to "uint16". This affects the following PDR definitions: 28.4 Numeric Sensor PDR 28.11 Numeric Effector PDR
1.1.2	2019-08-28	Errata update to correct field ordering in response message for FindPLDMEventLogEntry command
1.2.0	2019-09-09	Added Support for Redfish Device Enablement (DSP0218) Clarified Get - Set Sensor Threshold commands Added Compact Numeric Sensor PDR to simplify reporting of numeric data Extended PLDM event model to support synchronous (polled) events, and keepalive heartbeat timers Added PDR repository management commands to better support dynamic modifications to PDRs

3037
3038

3039

Bibliography

3040 DMTF DSP4004, *DMTF Release Process 2.4*,
3041 http://dmf.org/sites/default/files/standards/documents/DSP4004_2.4.pdf

3042