



Document Number: DSP0248

Date: 2022-10-26

Version: 1.2.2

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

Supersedes: 1.2.1

Document Class: Normative

Document Status: Published

Document Language: en-US

Copyright Notice

Copyright © 2009-2011, 2016, 2019, 2021, 2022 DMTF. All rights reserved.

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

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patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a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

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

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

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

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

CONTENTS

39	Foreword	10
40	Introduction.....	11
41	1 Scope	12
42	2 Normative references	12
43	3 Terms and definitions	13
44	4 Symbols and abbreviated terms.....	15
45	5 Conventions	15
46	6 PLDM for Platform Monitoring and Control version	16
47	7 PLDM for Platform Monitoring and Control overview	16
48	8 PDR architecture	18
49	8.1 General	18
50	8.2 Primary PDR Repository and Device PDR repositories	18
51	8.3 Use of PDRs	18
52	9 Entities.....	21
53	9.1 Entity Identification Information.....	22
54	9.2 Entity Type and Entity IDs	23
55	9.3 Entity Instance Numbers.....	24
56	9.4 Container ID	24
57	9.5 Use of Container ID in PDRs	24
58	10 PLDM associations.....	25
59	10.1 Association examples	25
60	10.2 Internal and External Associations	25
61	10.3 Sensor/Effecter to Entity associations	26
62	10.4 FRU Record Set to Entity associations.....	27
63	11 Entity Association PDRs.....	29
64	11.1 Physical-to-Physical containment associations	29
65	11.2 Entity identification relationships between PDRs	31
66	11.3 Linked Entity Association PDRs	32
67	11.4 Logical containment associations	33
68	11.5 Sensor/effecter associations with logical entities	34
69	11.6 Merged entity associations	35
70	11.7 Separation of logical and physical associations	37
71	11.8 Designing association PDRs for monitoring and control	37
72	11.9 Terminus associations	38
73	11.10 Interrupt associations.....	41
74	12 PLDM terminus.....	42
75	12.1 TIDs, PLDM Terminus Handles, and Terminus Locator PDRs	43
76	12.2 Requirements for unique TIDs.....	43
77	12.3 Terminus messaging requirements	43
78	12.4 Terminus Locator PDRs.....	43
79	12.5 Enumerating termini.....	44
80	13 PLDM events.....	45
81	13.1 PLDM Event Messages	46
82	13.2 PLDM Event Receiver.....	46
83	13.3 PLDM Event Logging	47
84	13.4 PLDM Event Log clearing policies.....	47
85	13.5 Oldest and newest log entries	48
86	13.6 Event Receiver Location.....	48
87	13.7 PLDM Event Log entry formats.....	48
88	13.8 PLDM Platform Event Entry Data format.....	49

89	13.9	OEM Timestamped Event Entry Data format	50
90	13.10	OEM Event Entry Data format	50
91	14	Discovery Agent	50
92	14.1	Assignment of TIDs and Event Receiver location	51
93	14.2	UUIDs for devices in hot-plug or add-in card applications.....	52
94	14.3	UID implementation	52
95	14.4	More than one terminus in a device.....	52
96	14.5	Examples of PDR and UUID use with add-in cards	52
97	15	Initialization Agent	55
98	15.1	General	55
99	15.2	PLDM and power state interaction.....	55
100	15.3	RunInitAgent command	55
101	15.4	Recommended Initialization Agent steps.....	56
102	16	Terminus and event commands	56
103	16.1	SetTID command	57
104	16.2	GetTID command.....	58
105	16.3	GetTerminusUID command	58
106	16.4	SetEventReceiver command	58
107	16.5	GetEventReceiver command.....	60
108	16.6	PlatformEventMessage command.....	61
109	16.7	PollForPlatformEventMessage command	62
110	16.8	EventMessageSupported Command.....	66
111	16.9	EventMessageBufferSize Command.....	67
112	16.10	eventData format for sensorEvent.....	68
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	86
134	18.5	RestoreSensorThresholds command	87
135	18.6	GetSensorHysteresis command	87
136	18.7	SetSensorHysteresis command	88
137	18.8	InitNumericSensor command	89
138	19	PLDM State Sensors.....	90
139	20	PLDM State Sensor commands.....	91
140	20.1	SetStateSensorEnables command.....	91
141	20.2	GetStateSensorReadings command	92
142	20.3	InitStateSensor command	94

143	21	PLDM effecters.....	95
144	21.1	PLDM State Effecters	95
145	21.2	PLDM Numeric Effecters	96
146	21.3	Effector semantics	96
147	21.4	PLDM and OEM effector semantic IDs.....	97
148	22	PLDM effector commands.....	97
149	22.1	SetNumericEffectorEnable command.....	97
150	22.2	SetNumericEffectorValue command.....	98
151	22.3	GetNumericEffectorValue command	99
152	22.4	SetStateEffectorEnables command.....	100
153	22.5	SetStateEffectorStates command.....	102
154	22.6	GetStateEffectorStates command	103
155	23	PLDM Event Log commands	104
156	23.1	GetPLDMEventLogInfo command	105
157	23.2	EnablePLDMEventLogging command.....	107
158	23.3	ClearPLDMEventLog command	107
159	23.4	GetPLDMEventLogTimestamp command	108
160	23.5	SetPLDMEventLogTimestamp command.....	108
161	23.6	ReadPLDMEventLog command	110
162	23.7	GetPLDMEventLogPolicyInfo command	112
163	23.8	SetPLDMEventLogPolicy command.....	114
164	23.9	FindPLDMEventLogEntry command	116
165	24	PLDM State Sets.....	118
166	25	Platform Descriptor Records (PDRs)	118
167	25.1	PDR Repository updates	118
168	25.2	Internal storage and organization of PDRs.....	118
169	25.3	PDR types.....	119
170	25.4	PDR record handles.....	119
171	25.5	Accessing PDRs	119
172	26	PDR Repository commands.....	119
173	26.1	GetPDRRepositoryInfo command	119
174	26.2	GetPDR command.....	121
175	26.3	FindPDR command.....	125
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	134
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.....	146
192	28.5	Numeric Sensor Initialization PDR.....	152
193	28.6	State Sensor PDR.....	153
194	28.7	State Sensor Initialization PDR.....	155
195	28.8	Sensor Auxiliary Names PDR	158
196	28.9	OEM Unit PDR.....	159
197	28.10	OEM State Set PDR	160

198	28.11 Numeric Effector PDR.....	162
199	28.12 Numeric Effector Initialization PDR.....	167
200	28.13 State Effector PDR.....	168
201	28.14 State Effector Initialization PDR.....	169
202	28.15 Effector Auxiliary Names PDR.....	172
203	28.16 OEM Effector Semantic PDR.....	173
204	28.17 Entity Association PDR.....	174
205	28.18 Entity Auxiliary Names PDR	175
206	28.19 OEM EntityID PDR.....	176
207	28.20 Interrupt Association PDR	177
208	28.21 Event Log PDR	178
209	28.22 FRU Record Set PDR	179
210	28.23 OEM Device PDR	180
211	28.24 OEM PDR	181
212	28.25 Compact Numeric Sensor PDR	182
213	28.26 Redfish Resource PDR.....	184
214	28.27 Redfish Entity Association PDR.....	187
215	28.28 Redfish Action PDR	188
216	29 Timing.....	190
217	30 PLDM Command numbers.....	190
218	ANNEX A (informative) Change log	192
219	Bibliography	194
220		

221 Figures

222	Figure 1 – PLDM used for access only	19
223	Figure 2 – PLDM with device PDRs.....	20
224	Figure 3 – PLDM with PDRs for subsystem.....	21
225	Figure 4 – Entity Identification Information.....	22
226	Figure 5 – Entity Identification Information format	22
227	Figure 6 – Entity Identification Information in a Numeric Sensor PDR	26
228	Figure 7 – Entity Identification Information in a FRU Record Set PDR.....	28
229	Figure 8 – Physical containment entity association PDR	30
230	Figure 9 – containerID relationships	31
231	Figure 10 – Entity identification relationship between PDRs	32
232	Figure 11 – Linked Entity Association PDRs	33
233	Figure 12 – Logical Containment PDR	34
234	Figure 13 – Sensor/effector to logical entity association	35
235	Figure 14 – Merged entity association PDR example.....	36
236	Figure 15 – Block diagram for merged entity association PDR example	37
237	Figure 16 – TID and PLDM Terminus Handle associations.....	39
238	Figure 17 – Block diagram of Terminus-to-Sensor associations	40
239	Figure 18 – Received interrupt association example	42
240	Figure 19 – Example of TID and PLDM Terminus Handle relationships	44
241	Figure 20 – Hot-plug add-in card with single PLDM terminus	53
242	Figure 21 – Hot-plug add-in card with multiple PLDM termini	54
243	Figure 22: Switching from asynchronous eventing to poll for an event with large data.....	64
244	Figure 23 – Numeric sensor threshold and hysteresis relationships	79
245	Figure 24 – Accuracy, tolerance, and resolution example	136

246	Figure 25 – Figuring resolution from the design	139
247		

248 Tables

249	Table 1 – PLDM monitoring and control data types.....	15
250	Table 2 – Parts of the Entity Identification Information format.....	22
251	Table 3 – Field & value descriptions for Entity Identification Information in a Numeric Sensor PDR.....	26
252	Table 4 – Field and value descriptions for Entity Identification Information in a FRU Record Set PDR.....	28
253	Table 5 – PLDM Event Log clearing policies	47
254	Table 6 – PLDM Event Log entry format.....	49
255	Table 7 – Platform Event Entry Data format	49
256	Table 8 – OEM Timestamped Event Entry Data format	50
257	Table 9 – OEM Event Entry Data format	50
258	Table 10 – Terminus and event commands.....	56
259	Table 11 – PLDM Event Types	57
260	Table 12 – GetTerminusUID command format.....	58
261	Table 13 – SetEventReceiver command format	59
262	Table 14 – GetEventReceiver command format.....	60
263	Table 15 – PlatformEventMessage command format.....	61
264	Table 16 – PollForPlatformEventMessage command format	65
265	Table 17 – EventMessageSupported command format	66
266	Table 18 – EventMessageBufferSize command format	68
267	Table 19 – sensorEvent class eventData format	68
268	Table 20 – effectorEvent class eventData format	71
269	Table 21 – redfishTaskExecutedEvent class eventData format	71
270	Table 22 – redfishMessageEvent class eventData format	72
271	Table 23 – pldmPDRRepositoryChgEvent class eventData format.....	73
272	Table 24 – pldmPDRRepositoryChgEvent changeRecord format.....	74
273	Table 25 – pldmMessagePollEvent class eventData format	74
274	Table 26 – heartbeatTimerElapsedEvent class eventData format	75
275	Table 27 – Threshold severity levels	77
276	Table 28 – Numeric Sensor commands.....	80
277	Table 29 – SetNumericSensorEnable command format	80
278	Table 30 – GetSensorReading command format	81
279	Table 31 – GetSensorThresholds command format.....	84
280	Table 32 – SetSensorThresholds command format	86
281	Table 33 – RestoreSensorThresholds command format	87
282	Table 34 – GetSensorHysteresis command format.....	88
283	Table 35 – SetSensorHysteresis command format	89
284	Table 36 – InitNumericSensor command format	90
285	Table 37 – State Sensor commands.....	91
286	Table 38 – SetStateSensorEnables command format.....	91
287	Table 39 – SetStateSensorEnables opField format.....	92
288	Table 40 – GetStateSensorReadings command format	93
289	Table 41 – GetStateSensorReadings stateField format	93
290	Table 42 – InitStateSensor command format	94

291	Table 43 – InitStateSensor initField format.....	95
292	Table 44 – Categories for effector semantics	96
293	Table 45 – State and Numeric Effector commands	97
294	Table 46 – SetNumericEffectorEnable command format	98
295	Table 47 – SetNumericEffectorValue command format	98
296	Table 48 – GetNumericEffectorValue command format	99
297	Table 49 – SetStateEffectorEnables command format.....	101
298	Table 50 – SetStateEffectorEnables opField format.....	101
299	Table 51 – SetStateEffectorStates command format	102
300	Table 52 – SetStateEffectorStates stateField format.....	102
301	Table 53 – GetStateEffectorStates command format	103
302	Table 54 – GetStateEffectorStates stateField format	103
303	Table 55 – PLDM Event Log commands	104
304	Table 56 – GetPLDMEventLogInfo command format.....	105
305	Table 57 – EnablePLDMEventLogging command format.....	107
306	Table 58 – ClearPLDMEventLog command format	108
307	Table 59 – GetPLDMEventLogTimestamp command format	108
308	Table 60 – SetPLDMEventLogTimestamp command format	109
309	Table 61 – ReadPLDMEventLog command format	111
310	Table 62 – PLDMEventLogData format	112
311	Table 63 – GetPLDMEventLogPolicyInfo command format	113
312	Table 64 – SetPLDMEventLogPolicy command format.....	115
313	Table 65 – FindPLDMEventLogEntry command format	117
314	Table 66 – PDR Repository commands.....	119
315	Table 67 – GetPDRRepositoryInfo command format	120
316	Table 68 – GetPDR command format.....	121
317	Table 69 – FindPDR command format	126
318	Table 70 – FindPDR Command Parameter Format Numbers.....	128
319	Table 71 – FindPDR command parameter formats	129
320	Table 72 – RunInitAgent command format	130
321	Table 73 – GetPDRRepositorySignature command format.....	131
322	Table 74 – sensorUnits enumeration	133
323	Table 75 – Common PDR header format	142
324	Table 76 – PDR Type Values.....	143
325	Table 77 – Terminus Locator PDR format	144
326	Table 78 – Numeric Sensor PDR format	146
327	Table 79 – Numeric Sensor Initialization PDR format	152
328	Table 80 – State Sensor PDR format	154
329	Table 81 – State Sensor possible states fields format.....	155
330	Table 82 – State Sensor Initialization PDR format	156
331	Table 83 – Sensor Auxiliary Names PDR format.....	158
332	Table 84 – OEM Unit PDR format.....	159
333	Table 85 – OEM State Set PDR format	161
334	Table 86 – OEM State Value Record format	162
335	Table 87 – Numeric Effector PDR format	163
336	Table 88 – Numeric Effector Initialization PDR format	167
337	Table 89 – State Effector PDR format	168
338	Table 90 – State Effector Possible States fields format.....	169

339	Table 91 – State Effector Initialization PDR format	170
340	Table 92 – Effector Auxiliary Names PDR format.....	172
341	Table 93 – OEM Effector Semantic PDR format.....	173
342	Table 94 – Entity Association PDR format.....	174
343	Table 95 – Entity Auxiliary Names PDR format	175
344	Table 96 – OEM EntityID PDR format	176
345	Table 97 - Interrupt Association PDR format	177
346	Table 98 – Event Log PDR format	178
347	Table 99 – FRU Record Set PDR format.....	180
348	Table 100 – OEM Device PDR format	181
349	Table 101 – OEM PDR format	181
350	Table 102 – Compact Numeric Sensor PDR format.....	182
351	Table 103 – Redfish Resource PDR format.....	184
352	Table 104 – Redfish Entity Association PDR format	187
353	Table 105 – Redfish Action PDR format	188
354	Table 106 – Monitoring and control timing specifications	190
355	Table 107 – Command numbers	190
356		
357		

Foreword

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

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

Acknowledgments

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

Editors:

- Bill Scherer – Hewlett Packard Enterprise
- Patrick Schoeller – Hewlett Packard Enterprise
- Tom Slaight – Intel Corporation

Contributors:

- Richelle Ahlvers – Broadcom Inc.
- Alan Berenbaum – SMSC
- Chris Bussan – Hewlett Packard Enterprise
- Patrick Caporale – Lenovo
- Phil Chidester – Dell Technologies
- Hoan Do – Broadcom Inc.
- Yuval Itkin – NVIDIA Corporation
- Ed Klodnicki – IBM
- John Leung – Intel Corporation
- Eliel Louzoun – Intel Corporation
- Balaji Natrajan – Microchip Technology Inc.
- Hemal Shah – Broadcom Inc.
- Tom Slaight – Intel Corporation
- Bob Stevens – Dell Technologies
- Supreeth Venkatesh – Arm Limited
- Harb Abdulhamid – Ampere Computing Inc.

388

Introduction

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

394 Document conventions

395 Typographical conventions

396 The following typographical conventions are used in this document:

- 397 • Document titles are marked in *italics*.
- 398 • Important terms that are used for the first time are marked in *italics*.
- 399

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

1 Scope

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

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

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

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

2 Normative references

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

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

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

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

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

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

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

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

3 Terms and definitions

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

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

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

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

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

3.1

contained entity

474 an entity that is contained within a container entity

3.2

container entity

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

3.3**container ID**

a numeric value that is used within Platform Descriptor Records (PDRs) to uniquely identify a container entity

3.4**containing entity**

an alternative way of referring to the container entity for a given entity

3.5**entity**

a particular physical or logical entity that is identified using PLDM monitoring and control data structures for the purpose of monitoring, controlling, or identifying that entity within the platform management subsystem, or for identifying the relationship of that entity to other entities that are monitored or controlled using PLDM monitoring and control

Examples of physical entities include processors, fans, power supplies, and memory chips. Examples of logical entities include a logical power supply (which may comprise multiple physical power supplies) and a logical cooling unit (which may comprise multiple fans or cooling devices).

3.6**Entity ID**

a numeric value that is used to identify a particular type of entity, but without designating whether that entity is a physical or logical entity

3.7**Entity Instance Number**

a numeric value that is used to differentiate among instances of the same type

For example, if two processor entities exist, one of them can be designated with instance number 1 and the other with instance number 2.

3.8**Entity Type**

a numeric value that identifies both the particular type of entity and whether the entity is a physical or logical entity

The Entity ID is a subfield of the Entity Type.

3.9**Platform Descriptor Record****PDR**

a set of data that is used to provide semantic information about sensors, effecters, monitored or controller entities, and functions and services within a PLDM implementation

PDRs are mostly used to support PLDM monitoring and control and platform events. This information also describes the relationships (associations) between sensor and control functions, the physical or logical entities that are being monitored or controlled, and the semantic information associated with those elements.

4 Symbols and abbreviated terms

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

4.1

CIM

Common Information Model

4.2

EID

Endpoint ID

4.3

IANA

Internet Assigned Numbers Authority

4.4

MAP

Manageability Access Point

4.5

MCTP

Management Component Transport Protocol

4.6

PDR

Platform Descriptor Record

4.7

PLDM

Platform Level Data Model

4.8

TID

Terminus ID

5 Conventions

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

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.

6 PLDM for Platform Monitoring and Control version

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

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

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

7 PLDM for Platform Monitoring and Control overview

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

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

- sensors and effecters

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

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

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

8 PDR architecture

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

8.1 General

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

This separation provides several benefits:

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

8.2 Primary PDR Repository and Device PDR repositories

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

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

8.3 Use of PDRs

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

8.3.1 PLDM for access only

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

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

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

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

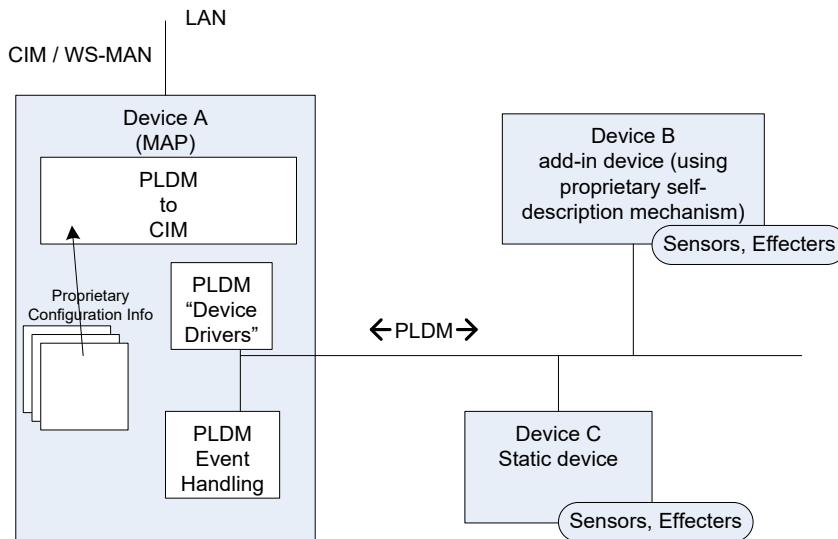


Figure 1 – PLDM used for access only

8.3.2 PLDM with PDRs for add-in devices

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

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

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

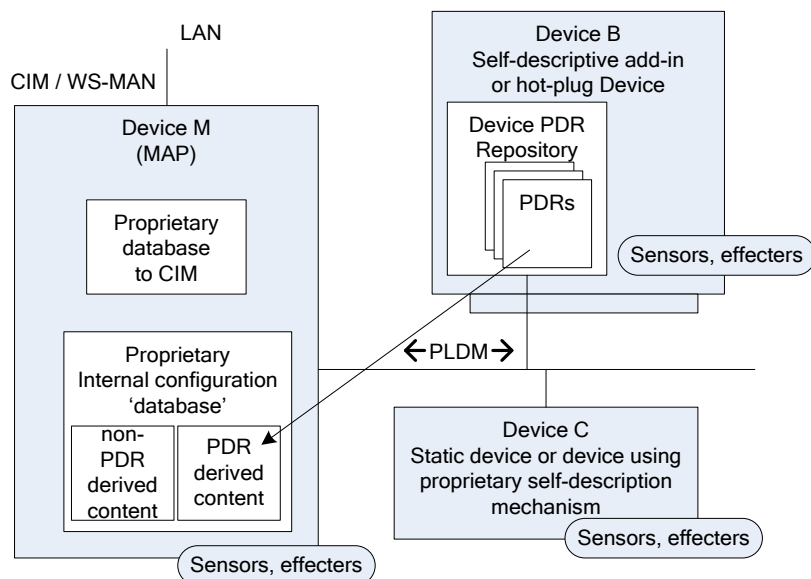


Figure 2 – PLDM with device PDRs

8.3.3 PLDM with Primary PDR Repository

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

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

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

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

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

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

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

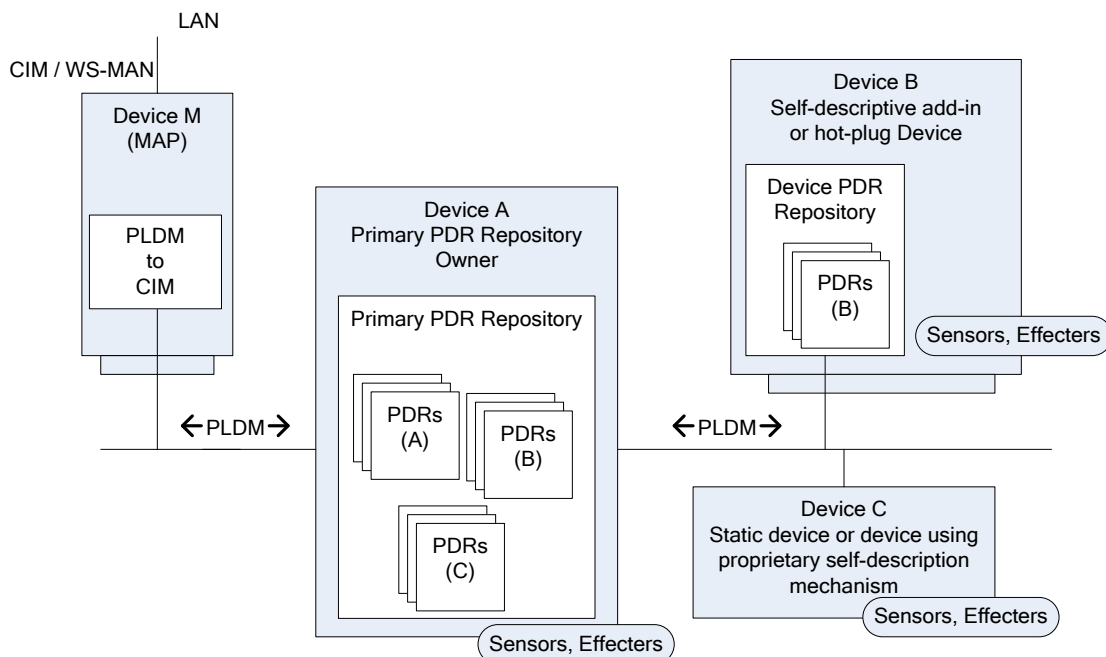


Figure 3 – PLDM with PDRs for subsystem

9 Entities

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

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

9.1 Entity Identification Information

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

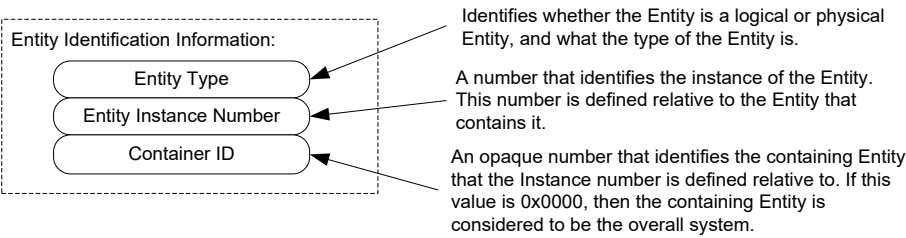


Figure 4 – Entity Identification Information

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

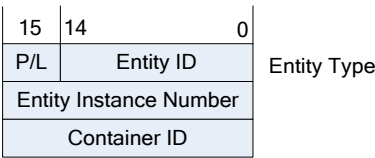


Figure 5 – Entity Identification Information format

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

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)

Part	Description
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.

9.2 Entity Type and Entity IDs

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

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

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

9.2.1 Vendor-specific (OEM) Entity IDs

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

9.2.2 Logical and physical entities

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

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

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

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

For example, consider a system with a motherboard that directly supports connectors for a redundant fan configuration. The fans would typically be individually replaceable, and the motherboard would be individually replaceable, but the "redundant fan subsystem" would not be. A user would not typically consider the combination of a motherboard and fans to be the definition of a physical redundant fan subsystem because the motherboard provides many other functions beyond those that are part of the

794 implementation of a redundant fan subsystem. The redundant fan subsystem does not have a distinct
795 physical boundary that would let it be replaced independently from other subsystems.

796 **9.3 Entity Instance Numbers**

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

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

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

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

810 **9.4 Container ID**

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

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

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

826 **9.5 Use of Container ID in PDRs**

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

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

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

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

10 PLDM associations

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

10.1 Association examples

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

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

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

10.2 Internal and External Associations

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

Therefore, Internal Associations require a common field to be defined among the elements that are associated with each other. The Internal Association mechanism is efficient, but not readily extensible, because a new type of association would typically require new fields to be defined and added to the

PDRs that are to be associated with one another, along with specifications that document how the field is used to form links to other records. Because the fields that support Internal Associations must be pre-defined as part of the PDR, Internal Associations are generally used only for the most fundamental and common types of associations. For other types of associations, a more generalized mechanism called "External Associations" is provided.

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

10.3 Sensor/Effector to Entity associations

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

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

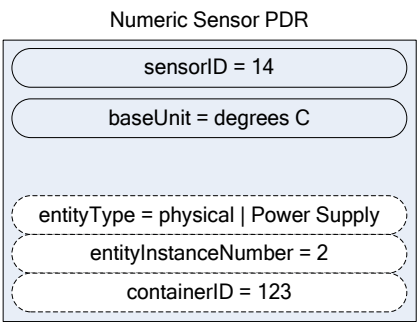


Figure 6 – Entity Identification Information in a Numeric Sensor PDR

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

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 PLDM terminus have unique sensorID or effectorID numbers. This field holds a value that is used in commands such as GetSensorReading to access the sensor or effector within the PLDM terminus. The sensorID is the PLDM terminus unique identifier used for

Field and value	Description
	accessing the sensor. The example shows that the value 14 would be used in commands to access the 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.

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

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

The information is based on the following fields:

container ID | entityType | entityInstanceNumber | sensorID | baseUnit

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

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

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

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

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

10.4 FRU Record Set to Entity associations

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

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

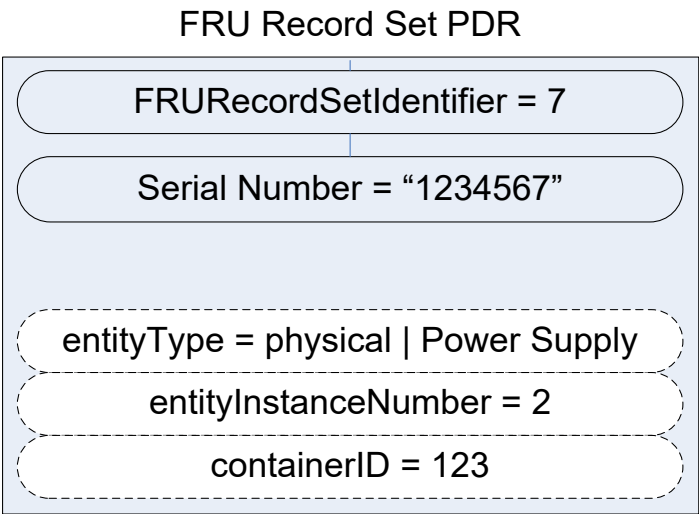


Figure 7 – Entity Identification Information in a FRU Record Set PDR

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

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

Field and value	Description
	containing entity (see 9.4 for more information). Association PDRs are described in detail in clause 11.

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

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

The information is based on the following fields:

container ID | entityType | entityInstanceNumber | Serial Number

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

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

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

"System Physical Power Supply 2 Serial Number"

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

11 Entity Association PDRs

Entity Association PDRs associate entities with one another.

11.1 Physical-to-Physical containment associations

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

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

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

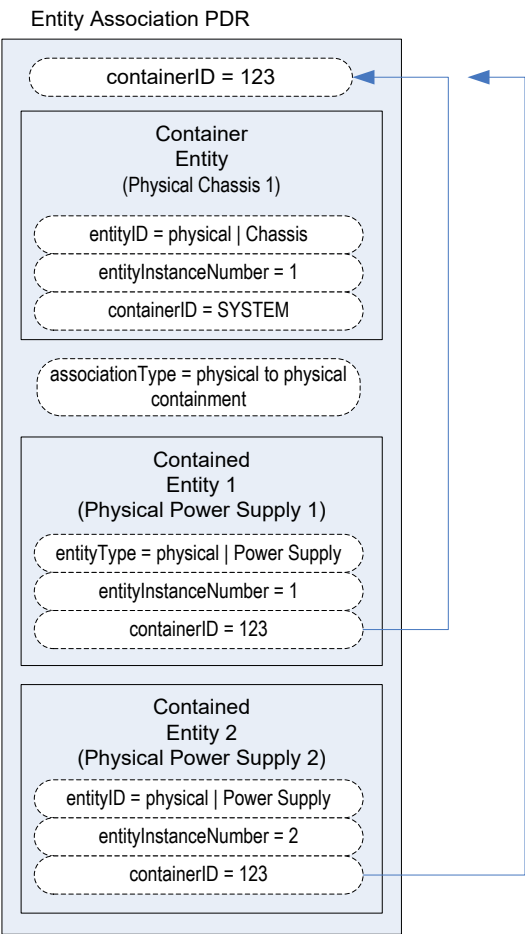
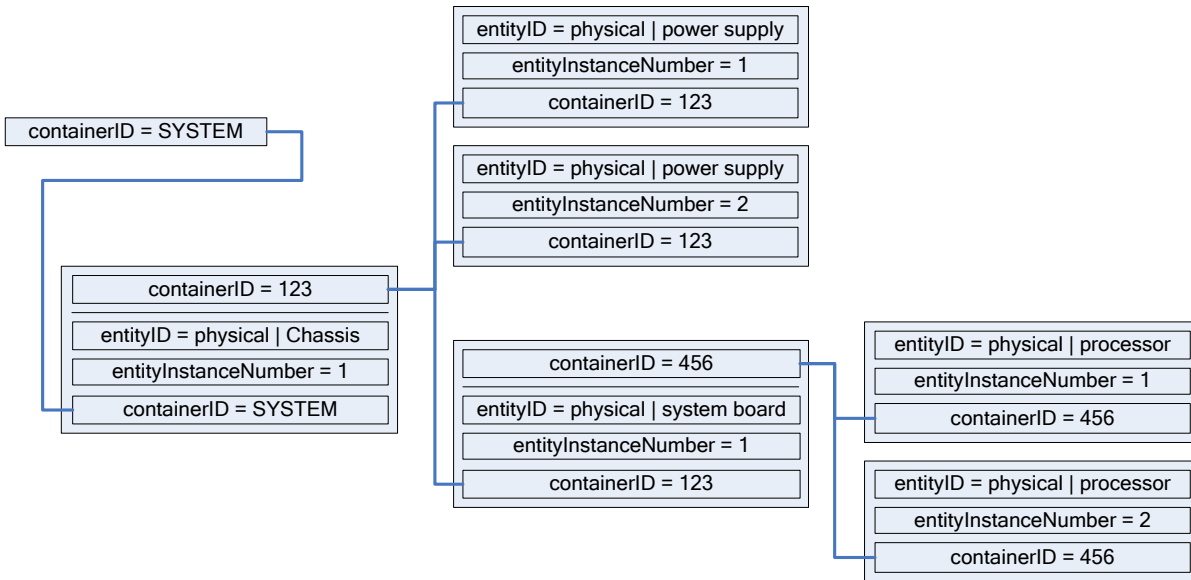


Figure 8 – Physical containment entity association PDR

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

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



977

978

Figure 9 – containerID relationships

979 11.2 Entity identification relationships between PDRs

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

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

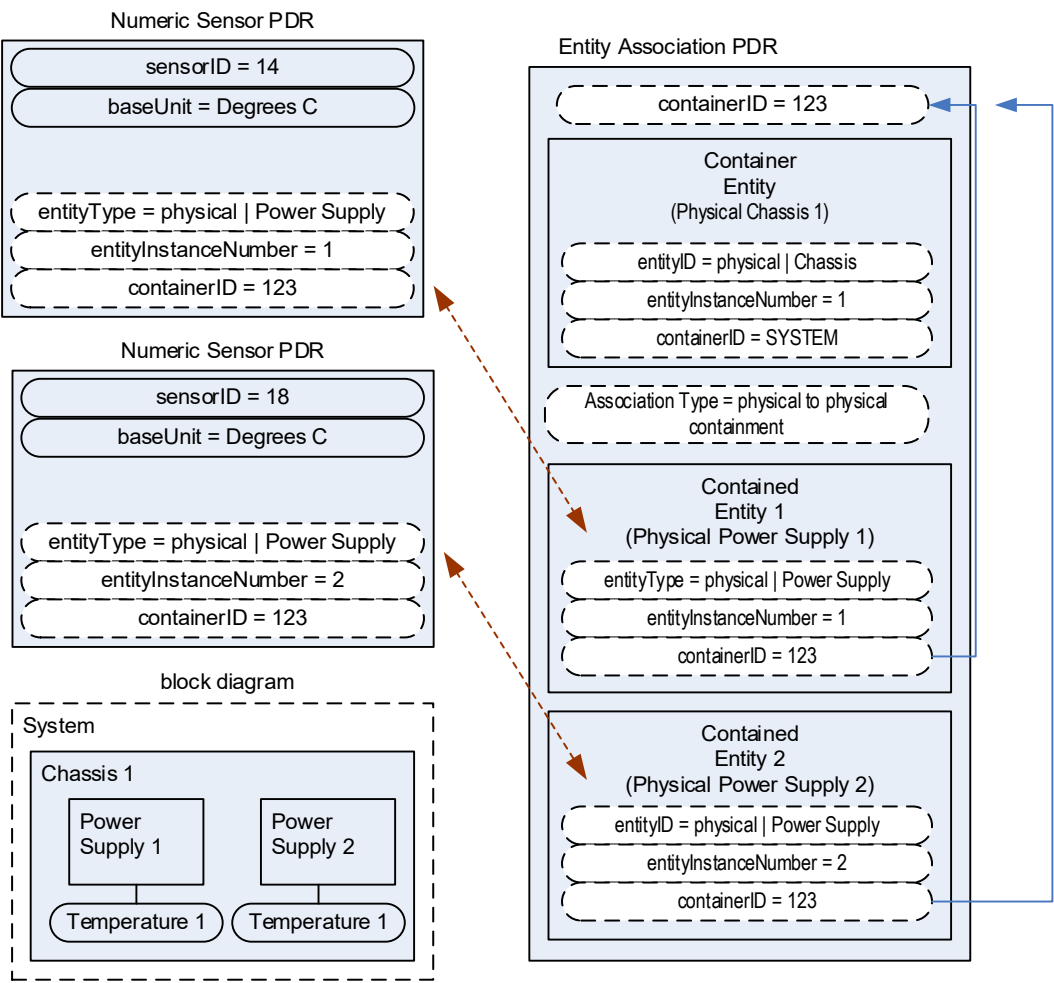


Figure 10 – Entity identification relationship between PDRs

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

11.3 Linked Entity Association PDRs

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

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

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

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

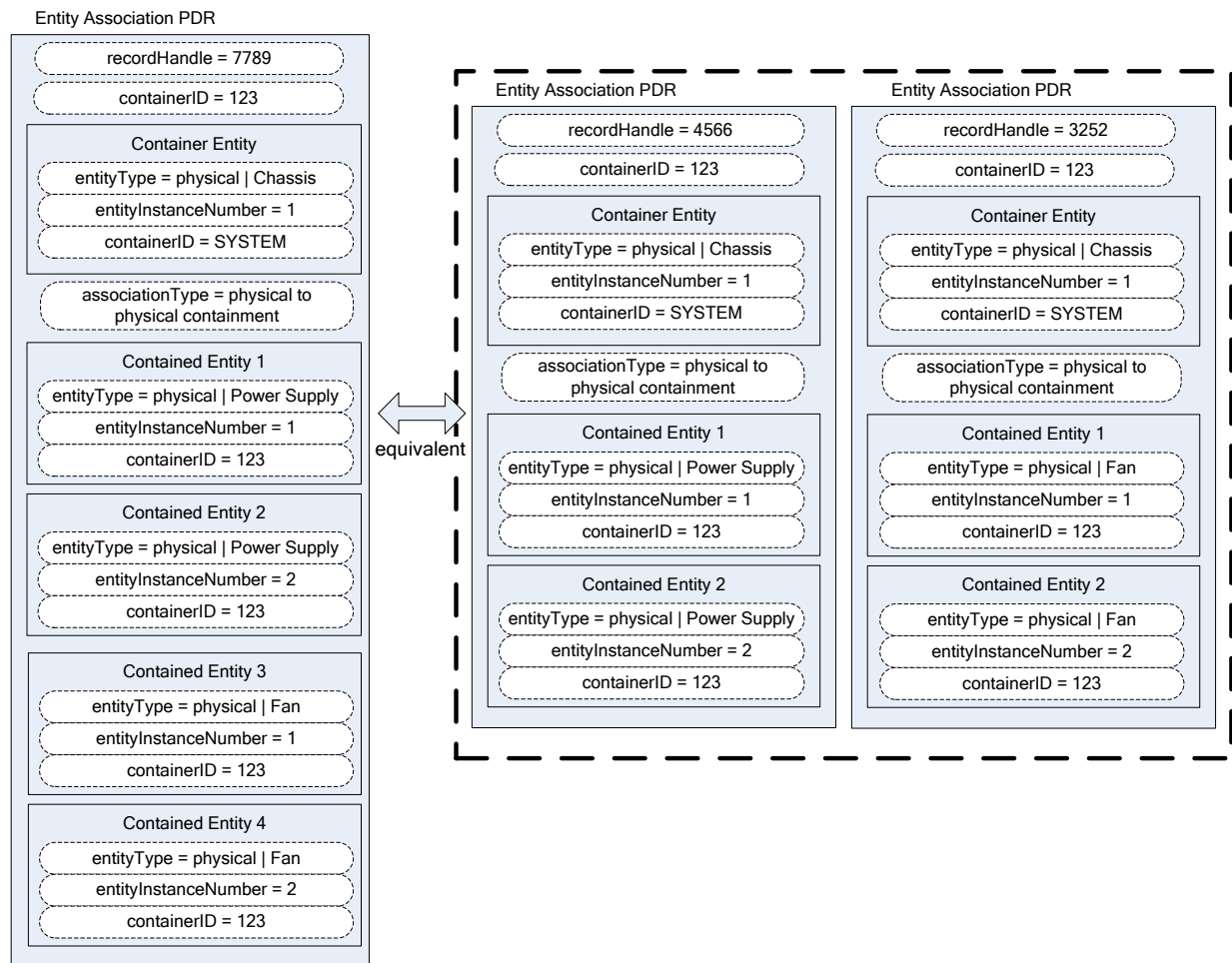


Figure 11 – Linked Entity Association PDRs

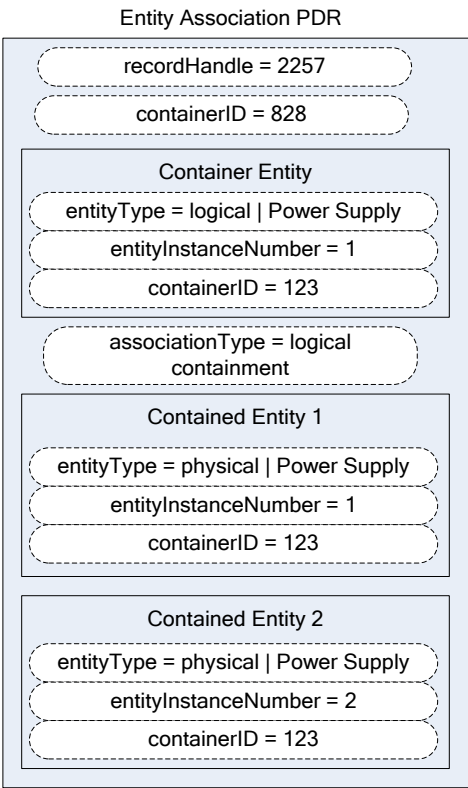
11.4 Logical containment associations

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

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

1021 The example PDR in Figure 12 shows a logical power supply 1 that contains physical power supply 1 and
1022 a physical power supply 2. In this example, the containerIDs in the enclosed Entity Identification
1023 Information do not reference the containerID of this overall PDR, but instead reference a container entity
1024 from a different PDR. This follows from the previous example where containerID 123 corresponds to
1025 physical chassis 1. The explanation for this is provided in 11.5.

1026 A logical containment association can have logical entities, physical entities, or both as contained entities.
1027 The container entity must always be defined as a logical entity.



1028

1029 **Figure 12 – Logical Containment PDR**

1030 **11.5 Sensor/effector associations with logical entities**

1031 Sensors and effectors can be associated with logical entities in the same way that they can be associated
1032 with physical entities. Figure 13 shows a state sensor that provides redundancy status and that has a
1033 sensor-to-entity association to logical power supply 1. Note that containerID 123 follows from the previous
1034 example where containerID 123 corresponds to physical chassis 1.

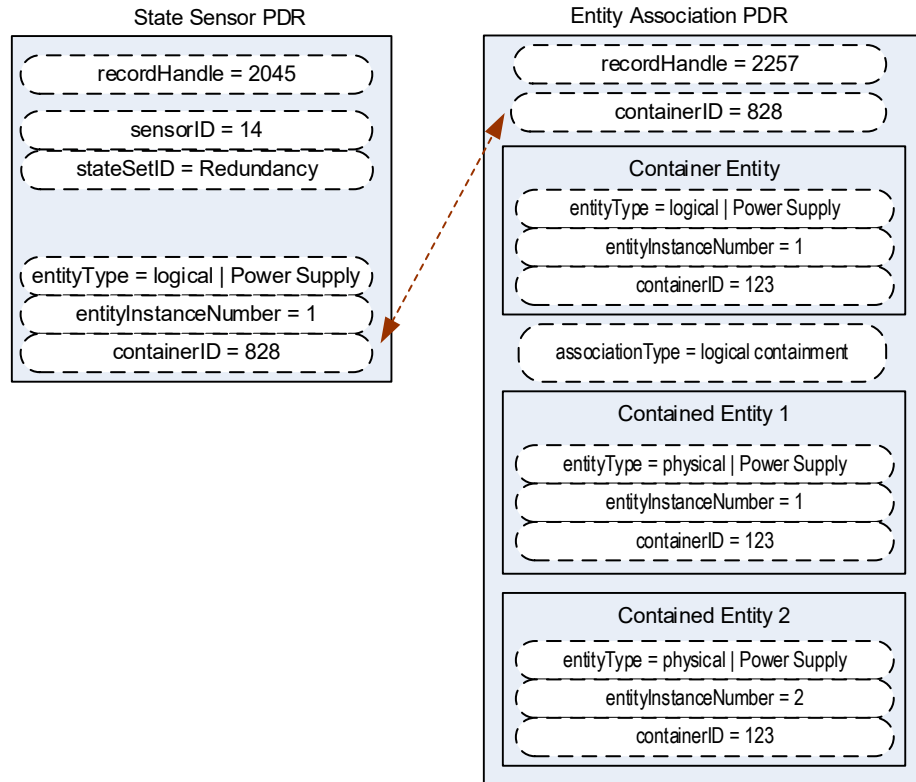


Figure 13 – Sensor/effector to logical entity association

11.6 Merged entity associations

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

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

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

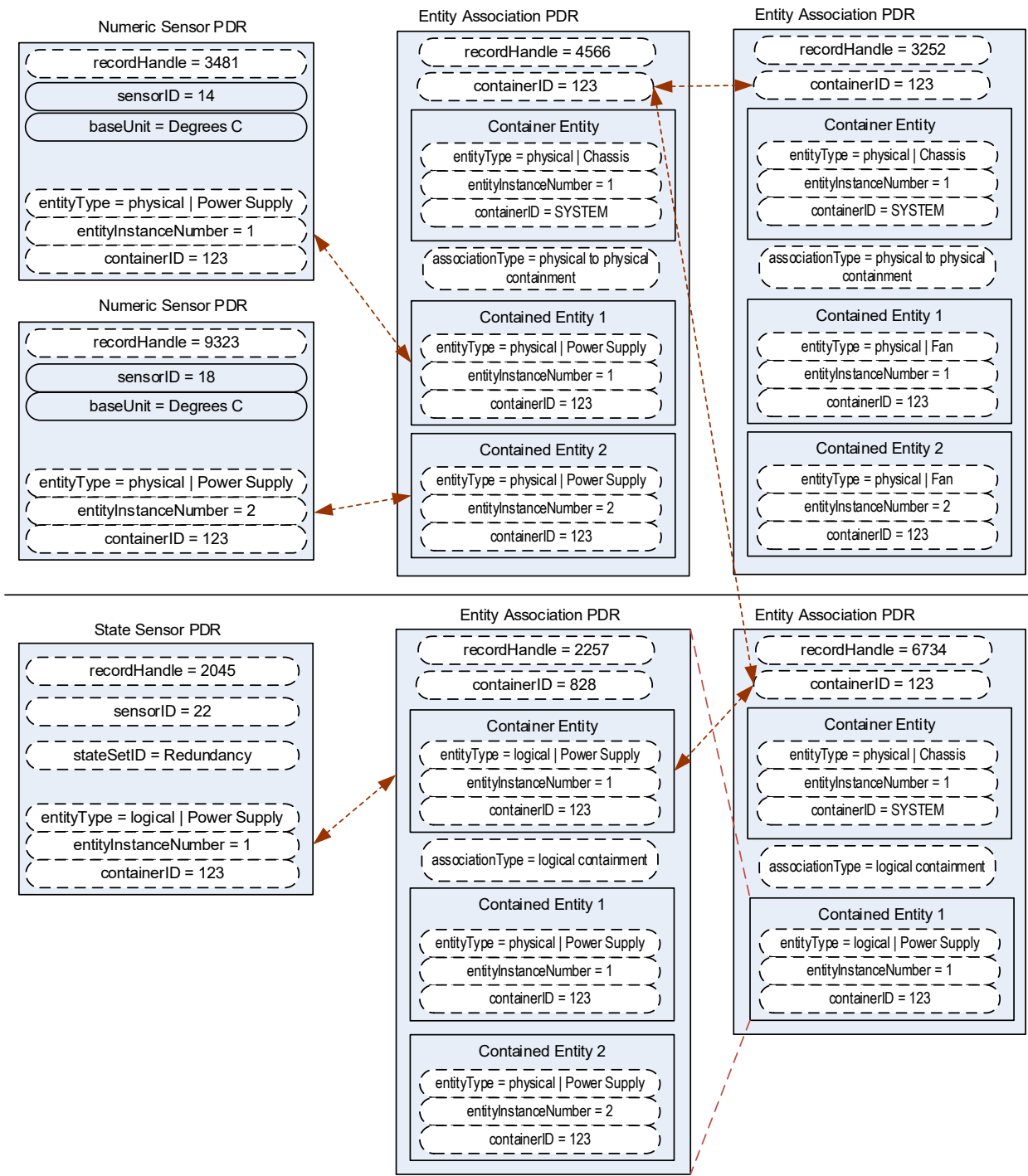


Figure 14 – Merged entity association PDR example

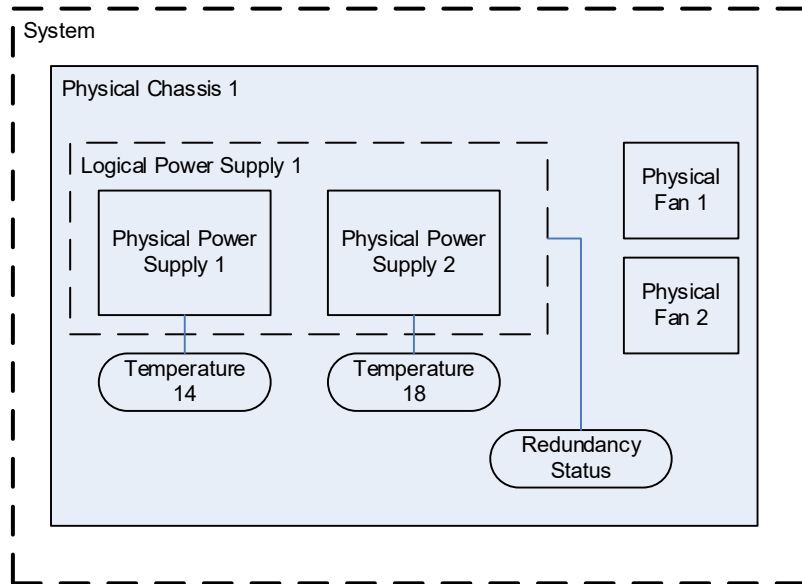


Figure 15 – Block diagram for merged entity association PDR example

11.7 Separation of logical and physical associations

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.

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.

11.8 Designing association PDRs for monitoring and control

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

- 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 is necessary to have a value that is more readily distinguishable as 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.

- 1076 4) Create the PDRs for any logical entities and set the containerID value for the containing entity to
1077 the containerID for the appropriate physical container entities.
- 1078 5) Create the Sensor PDR, Effector PDR, or other PDRs that reference those logical entities.

1079 11.9 Terminus associations

1080 Many PDRs that are related to monitoring and control include a value called the PLDM Terminus Handle.
1081 This is an opaque value that is used solely within the PDRs in a given repository as a means of identifying
1082 the records that are associated with a particular terminus. The Terminus ID (TID) is a value that is used
1083 with PLDM messaging as a way to identify a particular terminus. A PDR called the PLDM Terminus
1084 Locator PDR is used to bind the PLDM Terminus Handle and the TID for a given terminus.

1085 An overview of PLDM Terminus Handles and TIDs is given in 12.1. Figure 16 provides an illustration of
1086 the relationship of the PLDM Terminus Handle and TID and how they are used within the PDRs.

1087 The association of entities with sensors and effecters is independent of the terminus that provides access
1088 to the sensor or effector. Sensors and effecters are associated with the entity that is being monitored or
1089 controlled rather than the entity that is providing the PLDM terminus that is used to access the sensor or
1090 effector. For example, if a system board entity has a voltage sensor and a temperature sensor, the
1091 voltage sensor could be provided through one terminus and the temperature sensor through a different
1092 terminus. Both sensors would be associated with the same system board entity, however.

1093 Because Entity Association PDRs may have content in them that has associations with more than one
1094 terminus, the PLDM Terminus Handle is used to identify which terminus *provided* the PDR rather than
1095 which terminus *is associated with* the PDR. For example, this information can be used to identify when
1096 PDR information has been provided by an add-in card so that the PDRs can be updated if the add-in card
1097 is removed. In many applications, such as mapping PLDM to CIM, the PLDM Terminus Handle
1098 information in an Entity Association PDR can be ignored.

1099 Figure 16 also shows how the PLDMTerminusHandle field is used to identify which sensor PDRs are
1100 accessed through a particular terminus. The example shows two different termini providing sensors for
1101 the system. The terminus with TID 1 is bound to PLDMTerminusHandle 1000 using the Terminus Locator
1102 PDR with recordHandle 1776; the terminus with TID 2 is bound to PLDMTerminus Handle 1001 using the
1103 Terminus Locator PDR with recordHandle 1995.

1104 PLDMTerminusHandle 1000 is associated with the PDRs for two numeric temperature sensors that are
1105 then associated with physical power supplies 1 and 2. PLDMTerminusHandle 1001 is associated with a
1106 single redundancy state sensor that is associated with logical power supply 1. Figure 17 shows a block
1107 diagram of these relationships. Note that while this example shows different termini monitoring different
1108 entities, different termini can also provide sensors that monitor a common entity. For example, one
1109 terminus could provide voltage sensors for a processor while another terminus could provide a
1110 temperature sensor for the same processor.

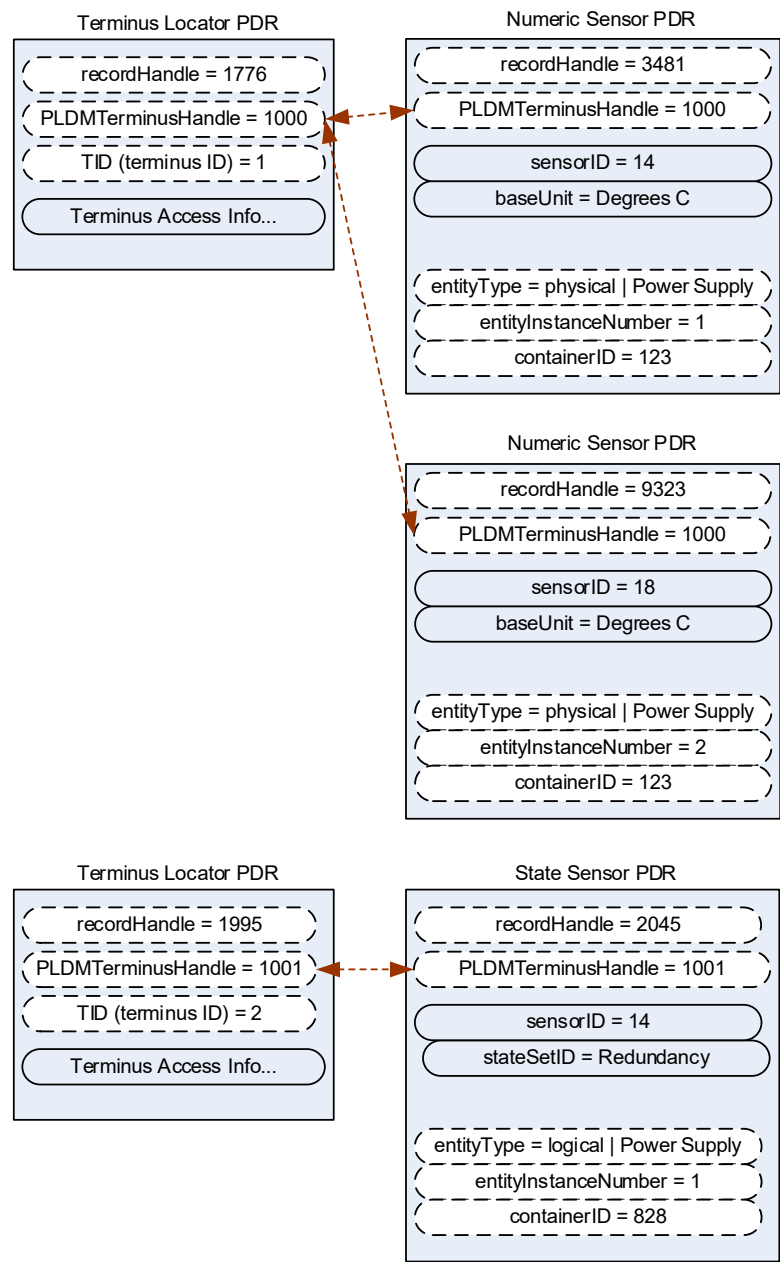


Figure 16 – TID and PLDM Terminus Handle associations

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

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

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

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

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

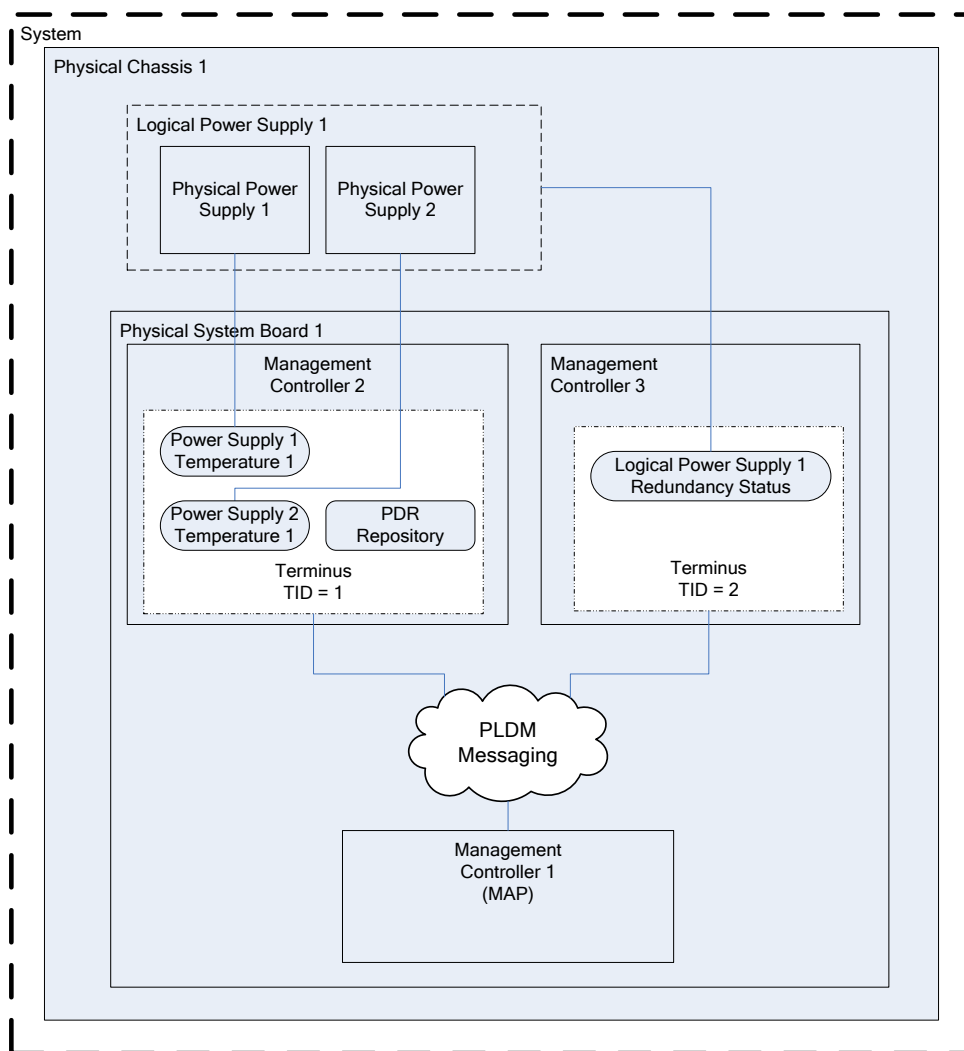


Figure 17 – Block diagram of Terminus-to-Sensor associations

11.10 Interrupt associations

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

11.10.1 Interrupt Association PDR

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

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

- **Received** interrupt associations identify when an interrupt target entity has received an interrupt from an interrupt source entity.
- **Requested** interrupt associations identify when an interrupt source has issued an interrupt request to an interrupt target entity.

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

11.10.2 Interrupt Association example

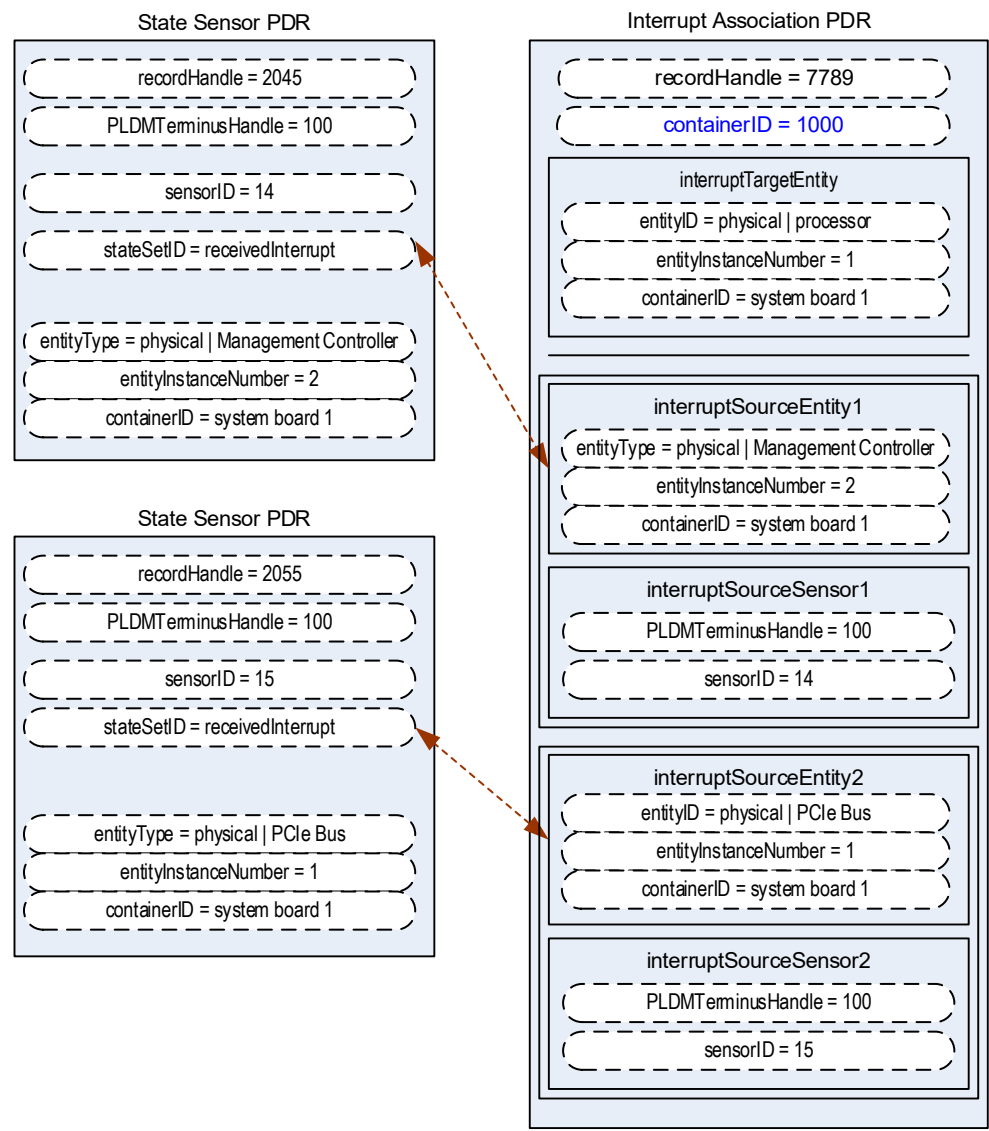
This clause presents an example of using an Interrupt Association PDR. In this example, processor 1 is the interrupt target entity that is associated with PCIe Bus 1 and Management Controller 2 as potential interrupt source entities. Management Controller 1 provides the implementation of two sensors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

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

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

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

1178 Entity Identification Information for the interrupt source entities. This additional information is required
1179 because a given source entity may be the source of more than one interrupt. The sensorID information
1180 provides the mechanism for differentiating different interrupts from the same interrupt source entity.



1181

1182 **Figure 18 – Received interrupt association example**

1183 **12 PLDM terminus**

1184 A PLDM terminus is the point of communication termination for PLDM messages and the PLDM functions
1185 associated with those messages. A terminus must be uniquely identifiable so that PLDM PDRs can
1186 associate semantic information with it. Additionally, a terminus must be identifiable when it generates

1187 asynchronous messages, such as event messages. This identification is accomplished through a value
1188 called the Terminus ID (TID).

1189 **12.1 TIDs, PLDM Terminus Handles, and Terminus Locator PDRs**

1190 The TID is primarily used in PLDM messages to identify which terminus generated an asynchronous
1191 message, such as an event message. The PLDM Terminus Handle is a value that is used within a PDR
1192 Repository to identify PDRs that are associated with a particular terminus. Thus, the PLDM Terminus
1193 Handle is defined only within the scope of a particular PDR Repository. A PDR called the Terminus
1194 Locator PDR is used to associate a TID with a Terminus Handle. The Terminus Locator PDR also
1195 includes information that describes how the terminus is accessed using PLDM messaging.

1196 **12.2 Requirements for unique TIDs**

1197 The assignment of unique TIDs to termini is required in the following situations:

- 1198 • Unique TIDs are required for implementations that use PDRs for describing sensors, effecters,
1199 and associations within and among termini.
- 1200 • Unique TIDs are required when an implementation exposes a PLDM Event Log in order to
1201 discriminate events from different termini when reading the log.

1202 **12.3 Terminus messaging requirements**

1203 PLDM termini that meet this specification must implement PLDM Request (command) and Response
1204 messages per [DSP0240](#). Additionally, a Management Controller that implements the Event Receiver
1205 function must be able to accept and process at least one Event Message request while it is processing
1206 other (non-Event Message) requests. Similarly, a device that generates Event Messages must be able to
1207 accept an incoming request while it is waiting for the response for the event message.

1208 It is recommended that a terminus can accept and track requests from multiple requesters if the terminus
1209 is used in an implementation where it is likely to receive simultaneous requests from multiple parties.

1210 **12.4 Terminus Locator PDRs**

1211 The Terminus Locator PDR forms the association between a TID and PLDM Terminus Handle for a
1212 terminus. The Terminus Locator PDR thus binds a given terminus and the semantic information that is
1213 provided through the PDRs for the terminus. Figure 19 illustrates the relationship between a TID and
1214 PLDM Terminus Handle.

1215 The Terminus Locator PDR also provides additional information about a terminus, such as how it can be
1216 accessed through PLDM messages (hence the name "Terminus Locator"), and whether the terminus and
1217 set of PDRs associated with that terminus should be considered present.

1218 If the terminus has a UID or UUID, the Terminus Locator PDR may also hold a copy of the UID/UUID
1219 value. This value provides an additional mechanism to help verify that the PDRs associated with the
1220 terminus are correct for the particular terminus instance.

1221 The relationship between the PDRs and PLDM Messaging to and from a given terminus is identified using
1222 the following data in the Terminus Locator PDR. (This information is expressed using multiple fields within
1223 the actual record format.)

- 1224 • The PLDM Terminus Handle is used to identify PDRs that are associated to a particular
1225 terminus. It is used only within the scope of a particular PDR Repository.
- 1226 • The TID identifies a terminus for PLDM messaging, particularly for identifying messages that
1227 come from a given terminus. A PLDM Terminus Locator PDR associates the TID with the PLDM
1228 Terminus Handle that is used for accessing the PDRs that are associated with the terminus.

- 1229 • The Terminus Access Info consists of a list of protocols and additional information, such as
1230 addressing, which enables a party to send PLDM messages to the terminus.

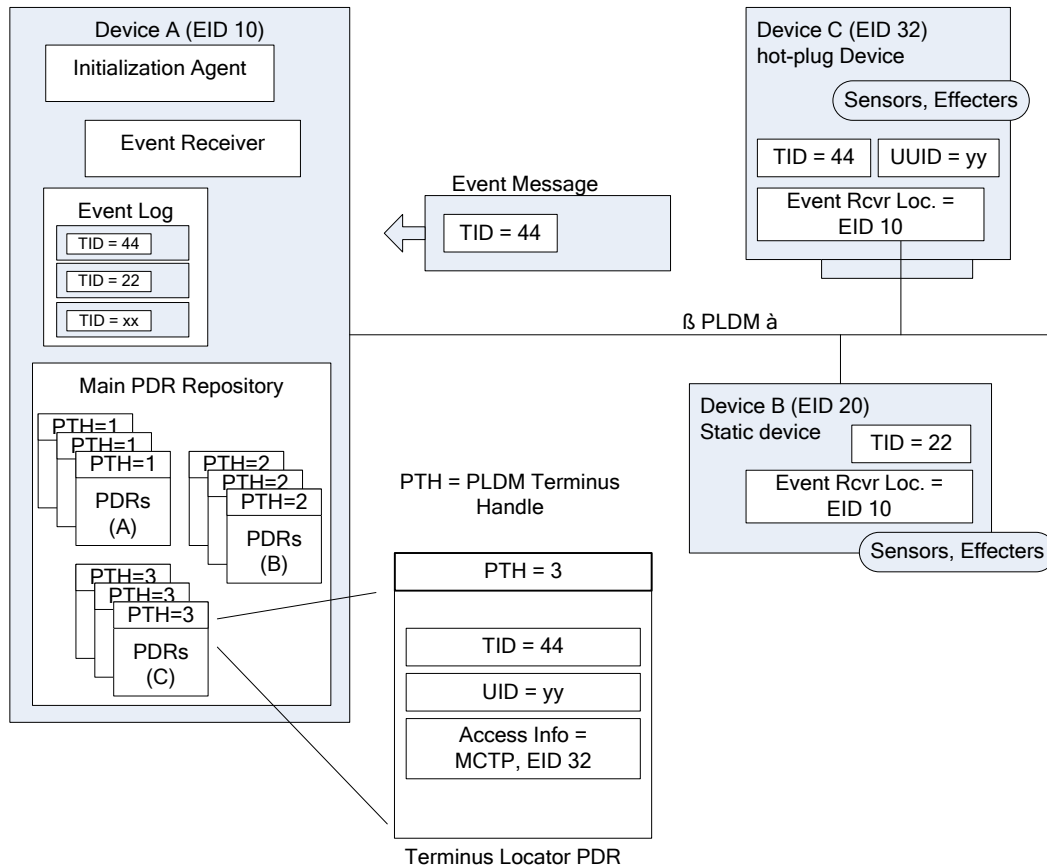


Figure 19 – Example of TID and PLDM Terminus Handle relationships

12.5 Enumerating termini

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

12.5.1 General

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

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

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

- A terminus must have a Terminus Locator PDR that is marked as valid in order to be considered present. Only one Terminus Locator PDR is allowed to be valid at a time for a given PLDM Terminus Handle within a PDR Repository. It is an error condition if multiple Terminus Locator PDRs exist and are simultaneously marked as valid for a given PLDM Terminus Handle.
- If the terminus has a sensor associated with it that reports Terminus State, the sensor must indicate that the terminus is present. Otherwise, the terminus and its associated PDRs should be ignored.
- If the terminus has a sensor associated with it that reports Terminus State and the Terminus State information cannot be accessed because the operationalState of the sensor is not "enabled", the terminus and its associated PDRs should be ignored.

12.5.2 Unlisted or absent termini

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

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

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

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

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

12.5.3 Terminus presence using Terminus State Sensors

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

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

13 PLDM events

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

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

1287 or entries in a CIM record log, while in other cases a PLDM event may be used solely to update CIM properties to
1288 eliminate or reduce polling by the MAP, or to report information about the internal health or operation of the PLDM
1289 subsystem that is not exposed through CIM.

1290 PLDM Events are between a PLDM terminus and the PLDM Event Receiver (such as a management
1291 controller). PLDM Events may be shared externally using the PLDM Event Log. The method to share the
1292 PLDM Event Log is outside the scope of this specification.

1293 **13.1 PLDM Event Messages**

1294 PLDM Event Messages are PLDM monitoring and control messages that are used by a PLDM terminus to
1295 synchronously or asynchronously report PLDM events to a central party called the PLDM Event Receiver.
1296 This specification version also adds a method to allow the event receiver to poll for events from the PLDM
1297 terminus event log.

1298 The PlatformEventMessage command supports multiple Event Data Classes.

1299 The PLDM terminus is expected to maintain an internal event message FIFO (queue) for both
1300 asynchronous transmission and polled message requests; All PLDM Event Messages are acknowledged
1301 by the PLDM Event Receiver using the command specific method. The number of entries in the PLDM
1302 terminus FIFO (queue) is implementation specific but should be sufficient to hold early events that occur
1303 before the PLDM Event Receiver configures the PLDM terminus for events. The FIFO should allow at
1304 least one event entry for each enabled sensor.

1305 The PLDM Event Receiver can only poll or accept PLDM Event Messages from the terminus after the
1306 terminus responds to the 16.4 SetEventReceiver command. The PLDM terminus may overwrite the oldest
1307 event (entry) or the oldest event for a specific sensor entry in the FIFO when the terminus (event) queue
1308 is full. Once a terminus transmits an event, the PLDM Event Receiver must acknowledge the event using
1309 the command specific acknowledgement. The acknowledged events are removed from the FIFO.

1310 There are two methods to transmit an event message to the event receiver:

1311 1. 16.6 PlatformEventMessage command

1312 This command allows the PLDM terminus to asynchronously transmit a PLDM event message to
1313 the established and designated PLDM Event Receiver. The Event Receiver acknowledges
1314 receiving the PLDM Event Message in the response to this command. DSP0240 (PLDM Base
1315 Specification) provides timing parameters in "Table 5 – Timing Specifications for PLDM
1316 Messages". The PLDM terminus is the Requester and shall retry sending this command "Number
1317 of request retries" (DSP0240, Table 5).

1318 2. 16.7 PollForPlatformEventMessage

1319 This command allows the designated PLDM Event Receiver to synchronously request (poll for) a
1320 PLDM terminus event message. The PLDM Event Receiver retrieves a single PLDM event
1321 message on each poll and should poll the terminus until the terminus indicates no more events.
1322 After the initial request (poll), the PLDM Event Receiver shall acknowledge the event returned on
1323 the next request (poll). The terminus may remove the event from the FIFO when the
1324 acknowledgement is received.

1325 **13.2 PLDM Event Receiver**

1326 The destination for event messages within PLDM is called the Event Receiver. The Event Receiver
1327 function is implemented by a PLDM terminus within the platform management subsystem. Multiple termini
1328 can send Event Messages to the Event Receiver function. The SetEventReceiver command is used to
1329 give the location of the Event Receiver function to termini that generate event messages.

1330 A PLDM Subsystem is defined as the collection of devices enumerated by the same PLDM initialization
1331 agent.

1332 A PLDM subsystem implementation can have only one PLDM Event Receiver function enabled at a given
1333 time. It is expected that typical implementations will always assign the same Event Receiver location.
1334 However, the location of the Event Receiver function is allowed to be changed during PLDM subsystem
1335 operation. For example, some implementations may do this to support a failover of the Event Receiver
1336 function, or to migrate it to a management controller that is hot plugged into the system, and so forth.

1337 13.3 PLDM Event Logging

1338 PLDM Event Logging defines an interface through which event messages that have been received at the
1339 Event Receiver can be saved in an area of storage called the PLDM Event Log for later retrieval. Event
1340 logging includes mechanisms for storing and time-stamping event records, determining characteristics of
1341 the log (such as its capacity), and reading and clearing the contents of the log.

1342 Additionally, "virtual" PLDM Event Messages may be internally generated within the terminus that is
1343 providing the PLDM Event Log function and directly logged without being received as PLDM Event
1344 Messages on any external interface.

1345 A PLDM terminus shall be tied to at most one PLDM Event Receiver and at most one PLDM Event Log
1346 function. The PLDM Event Log function is expected to be provided by a "time aware" management
1347 controller for the PLDM Subsystem. A simple PLDM terminus supporting a device or adapter should
1348 maintain an internal structure to support the 16.6 PlatformEventMessage command or the 16.7
1349 PollForPlatformEventMessage . The definition of this internal structure is implementation specific and
1350 outside the scope of this specification.

1351 Additional information about event logging is provided in clause 23.

1352 13.4 PLDM Event Log clearing policies

1353 The PLDM Event Log can use different policies for automatically clearing entries from the log (Table 5).
1354 The active policy is configured through the SetPLDMEventLogPolicy command. Refer to the specification
1355 of this command for policy support requirements.

1356 **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 Nth 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>

13.5 Oldest and newest log entries

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

13.6 Event Receiver Location

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

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

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

13.7 PLDM Event Log entry formats

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

1388

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.

1389 13.8 PLDM Platform Event Entry Data format

1390 Table 7 specifies the format used for the entryData field in PLDM Event Log entries that use the
1391 PLDMPlatformEvent value for the entryType field.

1392

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).

13.9 OEM Timestamped Event Entry Data format

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

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

13.10 OEM Event Entry Data format

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

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

14 Discovery Agent

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

1405 If the implementation is maintaining a Primary PDR Repository, the Discovery Agent may also be required
 1406 to automatically create or update PDRs to support devices such as hot-plug devices that may be
 1407 dynamically added or removed from the system. This includes the following actions:

- 1408 • creating records such as Terminus Locator PDRs
- 1409 • extracting Device PDR information and merging it into the Primary PDR Repository
- 1410 • updating associating records to link Device PDR information into the overall context of the
 1411 platform management subsystem

1412 Any OEM PDRs in the Device PDR information that are identified to be copied to the Primary PDR
 1413 Repository are also added to the Primary PDR Repository by the Discovery Agent.

1414 **14.1 Assignment of TIDs and Event Receiver location**

1415 Following are the support requirements for assignment of TIDs and the launching of the Initialization
 1416 Agent by a Discovery Agent within a PLDM implementation:

- 1417 • All termini must support the SetTID command.
- 1418 • All termini that generate PLDM Event Messages shall support the SetEventReceiver command.
 1419 Termini that do not generate PLDM Event Messages are not required to support the
 1420 SetEventReceiver command. Those termini, however, that support “Polled Events” shall support
 1421 the SetEventReceiver command.
- 1422 • The Discovery Agent function is responsible for discovering termini and assigning them unique
 1423 TID values. (A default TID setting may be preconfigured for a PLDM terminus if the terminus is
 1424 statically configured into the platform. This setting must be able to be overridden using the
 1425 SetTID command.)
- 1426 • The Initialization Agent function is responsible for initializing PLDM sensors and effecters and
 1427 setting Event Receiver location information into the termini. (A default Event Receiver setting
 1428 may be preconfigured for a PLDM terminus if the terminus is statically configured into the
 1429 platform. This setting must be able to be overridden using the SetEventReceiver command.)
 1430 The Initialization Agent function is described in more detail in clause 15.
- 1431 • When PDRs are used, the Initialization Agent is also responsible for maintaining corresponding
 1432 Terminus Locator PDR information.
- 1433 • A terminus must have its Event Receiver information set before it can begin to issue PLDM
 1434 Event Messages.
- 1435 • A terminus that has standby power should retain its TID and Event Receiver settings. When the
 1436 terminus comes back online, it can use that information for event messaging without requiring
 1437 Event Receiver reinitialization.
- 1438 • A terminus should retain its TID and Event Receiver settings during a given PLDM subsystem
 1439 operation.
- 1440 • Termini that are to be rediscovered (that is, termini that are not statically configured into the
 1441 system and may lose PLDM communication temporarily, which might occur in different platform
 1442 power states) must have a separate unique and persistent ID that can be associated with the
 1443 terminus. For example, if a terminus is hot-plug, it should have a universally unique ID (UUID).
- 1444 • TIDs are not required to persist or remain constant across PLDM subsystem restarts, unless the
 1445 system is using PDRs or exposes a PLDM Event Log. In such cases, TIDs must be persistently
 1446 stored by the termini or reassigned to the same value by the Discovery Agent function.
- 1447 • A MAP or other entity that is accessing a PLDM subsystem should not cache TIDs because
 1448 TIDs might change if the PLDM subsystem is reset or reinitialized.

- 1449 • Termini on hot-plug cards must have a UUID or be associated with a terminus on the same card
1450 that has a UUID.
- 1451 • Implementations that do not use PDRs can assign TIDs in any manner, including not assigning
1452 them at all. In this case, the implementation must define its own mechanisms for identifying and
1453 tracking termini and event messages from termini.

1454 14.2 UUIDs for devices in hot-plug or add-in card applications

1455 If the device is intended to be used on an add-in or hot-plug card, it may be required to support a
1456 universally unique ID (UUID) depending on higher-level system requirements or initiatives. In general,
1457 add-in cards that plug into standardized I/O connections and are used in multiple vendor systems, such
1458 as PCIe add-in cards, are required to use UUIDs so that multiple instances of the same card can be
1459 detected.

1460 14.3 UID implementation

1461 If a terminus is required to have a unique ID (UID), how the UID is implemented depends on the
1462 component and how the device manufacturer intends the device to be used in a system. For example, it
1463 is the device manufacturer's choice whether the entire UID must be configured by the system integrator
1464 after purchasing the device, or a number of preconfigured UIDs in the device are selectable by a pin or
1465 nonvolatile configuration selection, or the UID is permanently embedded in the device. Typically, each
1466 device will have fuses, PROM, EPROM/EEPROM, or some other nonvolatile mechanism for holding the
1467 unique ID that is configured either during device manufacture or when the device is integrated into a
1468 system.

1469 14.4 More than one terminus in a device

1470 The Terminus Locator PDR contains a containerEntity field that can be used to identify the entity that
1471 contains the terminus. This field provides the mechanism to identify when multiple termini are within the
1472 same device or are located within the same entity.

1473 14.5 Examples of PDR and UUID use with add-in cards

1474 Figure 20 and Figure 21 present examples of how Device PDRs, UUIDs, and Terminus Locator PDRs
1475 work together to identify PLDM termini on add-in cards, such as hot-plug add-in cards, that may be
1476 dynamically inserted or removed during PLDM subsystem operation. Both examples illustrate MCTP-
1477 based implementations. However, the approach may be extrapolated to other transport types.

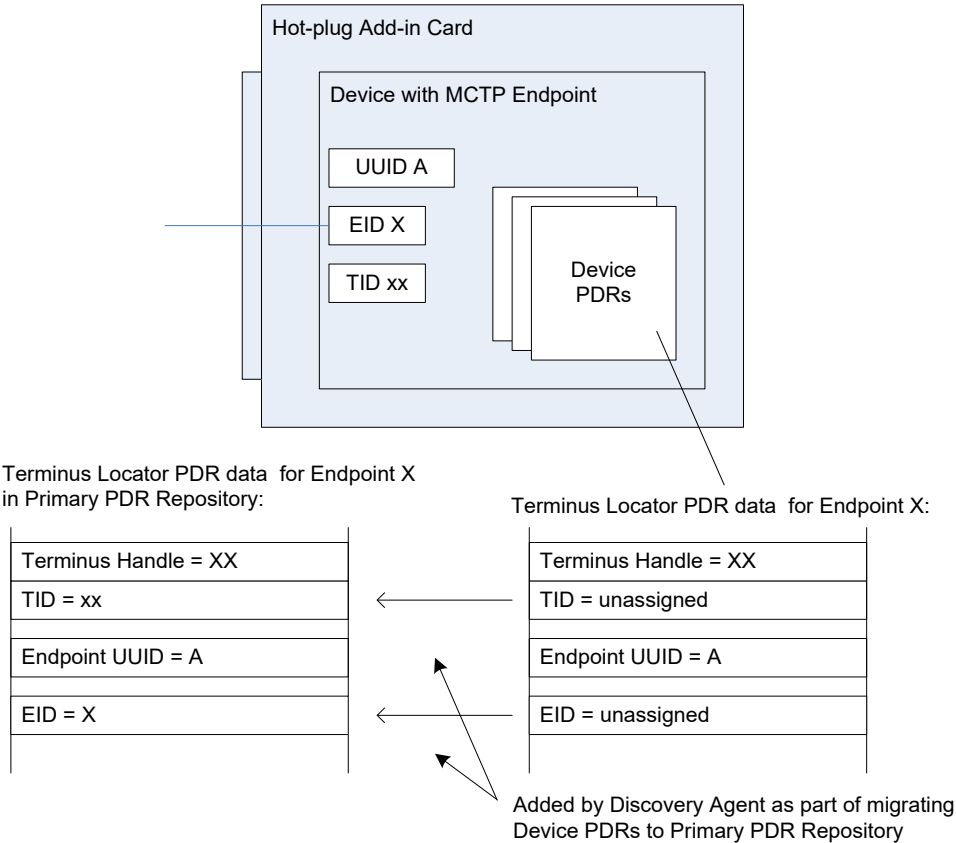


Figure 20 – Hot-plug add-in card with single PLDM terminus

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

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

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

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

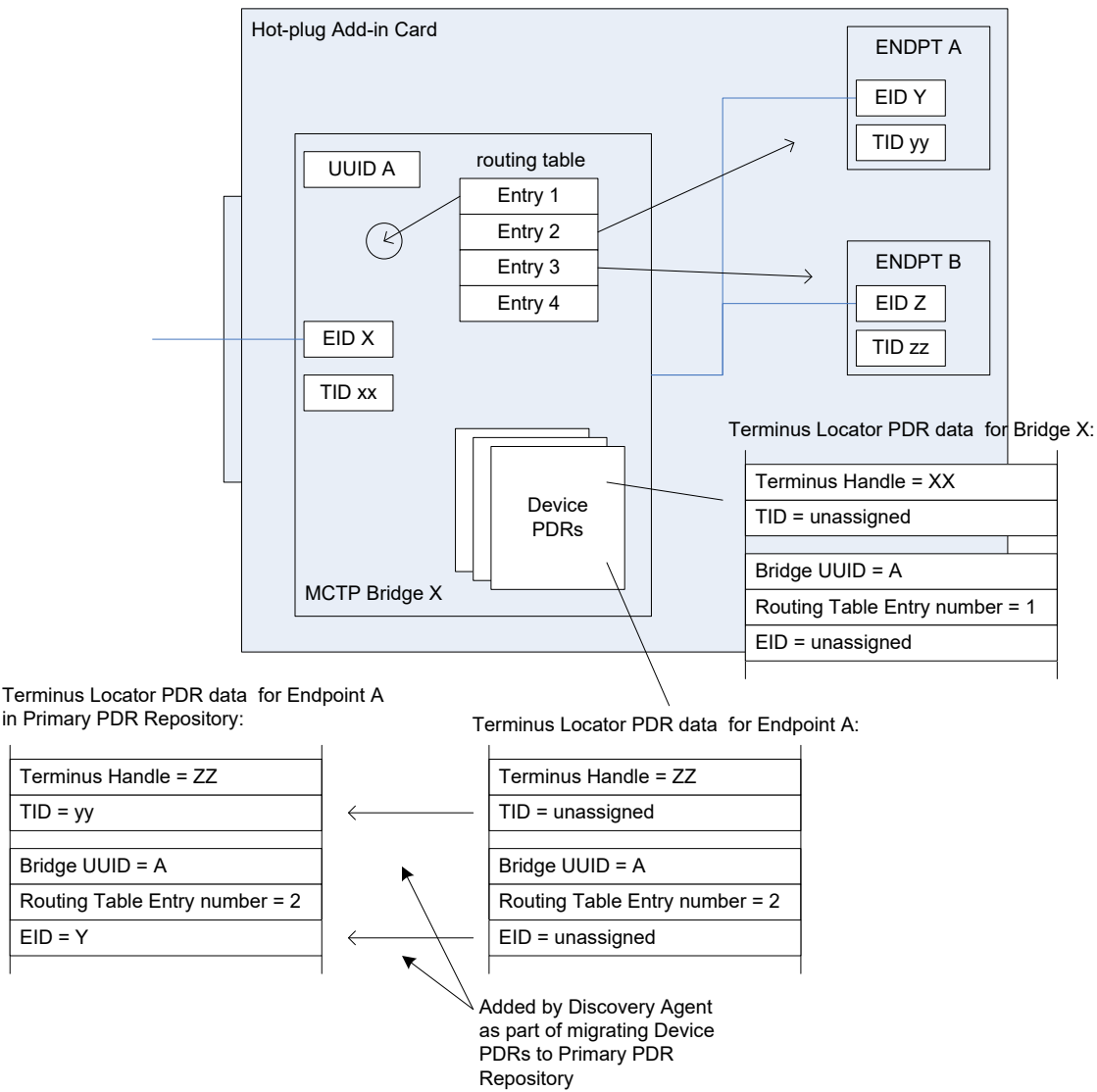


Figure 21 – Hot-plug add-in card with multiple PLDM termini

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

1510 PLDM subsystem operation. This approach eliminates the need for Endpoints A and B to have their own
1511 support for UUIDs.

1512 **15 Initialization Agent**

1513 This clause describes the role and operation of the Initialization Agent function in a PLDM subsystem that
1514 uses PDRs.

1515 **15.1 General**

1516 PLDM sensors are not required to completely self-initialize and enable themselves upon PLDM
1517 subsystem startup or upon power state changes of the device that is hosting the sensor. Thus, low-cost
1518 devices are not required to have nonvolatile configuration resources. Additionally, the mechanism
1519 provides options for overriding default configurations of sensors and event generation.

1520 The Initialization Agent is a function that initializes message generation and sensor configuration as
1521 described by Sensor Initialization PDRs. The Initialization Agent function normally runs whenever the
1522 platform management subsystem is first powered up, upon system Hard and Soft Resets, and on certain
1523 other transitions. Fields in the Sensor Initialization PDRs indicate the system transitions on which a given
1524 sensor is initialized.

1525 The Initialization Agent is also responsible for setting the Event Receiver Location information and
1526 enabling event message generation.

1527 The Sensor Initialization PDRs hold information that describes the default threshold values, states, and
1528 event generation settings for sensors that are initialized by the Initialization Agent function. Sensor
1529 Initialization PDRs are required only for sensors that are initialized by the Initialization Agent. Sensors that
1530 are self-initializing or are initialized through some mechanism that is outside the PLDM specifications do
1531 not need Sensor Initialization PDRs.

1532 The Initialization Agent function thus eliminates the need for all sensors to retain their own nonvolatile
1533 storage for their default settings, and also provides a mechanism to retrigger any events that may have
1534 been transmitted before the Event Receiver function was ready to accept them.

1535 Only one Initialization Agent function is supported within a given PLDM subsystem. The Initialization
1536 Agent shall be implemented behind the same terminus that provides the Primary PDR Repository for the
1537 PLDM subsystem.

1538 **15.2 PLDM and power state interaction**

1539 The Initialization Agent may need to reinitialize certain sensors or termini as the result of a change of
1540 system power state. An implementation should avoid requiring the Initialization Agent to execute because
1541 of low-latency power state transitions, such as transitions between ACPI S0 and S1, or S1 and S2 states.
1542 The implementation should instead ensure that termini retain their settings across low-latency power state
1543 transitions.

1544 The Sensor Initialization PDRs include a field that tells the Initialization Agent upon which system
1545 transitions a given sensor should be initialized.

1546 **15.3 RunInitAgent command**

1547 PLDM does not specify a particular mechanism for an implementation to use to detect when to run the
1548 Initialization Agent function. For example, it does not specify how a management controller would detect a
1549 system hard reset or power-up transition. In some implementations, it will be useful to have another
1550 management controller, system firmware, or another entity decide that the Initialization Agent should run.
1551 For example, system firmware may decide that the Initialization Agent should be run after a BIOS update.

1552 To enable this, PLDM defines a RunInitAgent command that can be used to launch the Initialization Agent
 1553 “on demand.” The command includes a parameter that can select a subset of Sensor Initialization PDRs
 1554 to be used.

1555 15.4 Recommended Initialization Agent steps

1556 The following presents an outline of the steps for an Initialization Agent in a system implementation that
 1557 includes Initialization PDRs.

- 1558 1) Stop the Event Receiver function from accepting events received from any interface but the system
 1559 (host) interface.
- 1560 2) Scan the PDR Repository for Terminus Locator PDRs. Collect a list of valid termini.
- 1561 3) For each terminus in the list, perform the following actions:
 - 1562 a) Turn off Event Generation by using the SetEventReceiver command. If a terminus responds to
 1563 the SetEventReceiver command, add the terminus to a list of termini to have events re-enabled
 1564 later.
 - 1565 b) Use the GetTID command to determine whether the terminus has a TID. If so, leave that value
 1566 unchanged unless it is already assigned to another terminus. If not, use the SetTID command to
 1567 assign a TID to the terminus.
 - 1568 c) Scan the PDR Repository for Initialization PDRs (for example, numeric sensor/effector
 1569 initialization PDRs or state sensor/effector initialization PDRs) that are associated for the
 1570 terminus. For each PDR that is found, perform the following actions:
 - 1571 – Set the sensor type, sensor thresholds, and hysteresis as directed by the PDR using the
 1572 SetSensorThresholds and SetSensorHysteresis commands.
 - 1573 – Use the appropriate enabling command (for example, SetNumericSensor Enables if the
 1574 sensor is a numeric sensor) to enable scanning and event generation per the PDR.
- 1575 4) Enable the Event Receiver function to accept or poll for event messages.
- 1576 5) For each terminus with a Terminus Locator PDR, enable synchronous or asynchronous event
 1577 message generation using the SetEventReceiver command or leave it disabled (This is done at the
 1578 discretion of the Management Controller.) For each of these termini, configure an event message
 1579 transfer size via the EventMessageBufferSize command.

1580 16 Terminus and event commands

1581 This clause describes the commands that are used by PLDM termini that implement PLDM monitoring
 1582 and control as defined in this specification. The command numbers for the PLDM messages are given in
 1583 clause 30.

1584 If a PLDM terminus is implemented to provide access to any of the capabilities of this specification, the
 1585 Mandatory/Conditional (M/C) requirements shown in Table 10 apply.

1586 **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

^[1] See 16.3.

^[2] Support for at least one of PlatformEventMessage or PollForPlatformEventMessage is mandatory for termini that generate PLDM Event Messages.

^[3] Sending the SetEventReceiver command is Mandatory for termini that implement the Initialization Agent function.

^[4] Mandatory for termini that generate redfishTaskExecutedEvent, redfishMessageEvent, or heartbeatTimerElapsedEvent class PLDM Event Messages.

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

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

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

16.1 SetTID command

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

16.2 GetTID command

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

16.3 GetTerminusUID command

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

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

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

Table 12 – GetTerminusUID command format

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

16.4 SetEventReceiver command

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

The event originator (terminus) will receive the request to enable legacy asynchronous event message, enable polling of event messages or disable all event message generation. This command permits only one eventMessageGlobalEnable enumeration and is superseded by subsequent invocations of this command. This specification has added additional completion codes to allow the terminus to indicate its

1627 capabilities. While this causes the requestor to reiterate the command to determine support, the method
 1628 preserves backward compatibility to previous specifications.

1629 Table 13 describes the format of the command.

1630 **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	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. 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. When enableAsync is chosen, the Event Receiver may also need to poll for large multipart event messages.
	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	transportProtocolType 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 59 completionCode returned if the transportProtocolType is incorrect.	
varies	eventReceiverAddressInfo 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	

	<p>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>

16.5 GetEventReceiver command

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

Table 14 – GetEventReceiver command format

Type	Request data
–	none
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES }</p>
enum8	<p>transportProtocolType</p> <p>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.</p>

varies	<p>eventReceiverAddress</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 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.</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 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.</p> <p>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.</p>
--------	--

1635

16.6 PlatformEventMessage command

1636 PLDM Event Messages are sent as PLDM request messages to the Event Receiver using the
1637 PlatformEventMessage command. Because PLDM requests have associated responses, this approach
1638 provides a positive acknowledgement that the event message was received. Table 15 describes the
1639 format of the command.

1640 When the terminus supplies a pldmMessagePollEvent, this indicates to the Event Receiver that the event
1641 data is large and must be retrieved via a series of multi-part transfers using the
1642 PollForPlatformEventMessage command. An example of this message flow may be found in clause 16.7.

1643 The formatVersion field shall be fixed at 0x01 for this format.

1644

Table 15 – PlatformEventMessage 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 the format detailed in this specification.</p>
uint8	<p>TID</p> <p>Terminus ID for the terminus that originated the event message</p>
uint8	<p>eventClass</p> <p>The class of event being sent. See Table 11 for a list of event types.</p>
var	<p>eventData</p> <p>Event data based on the eventClass</p>
Type	Response data
enum8	<p>completionCode</p> <p>value: { PLDM_BASE_CODES, UNSUPPORTED_EVENT_FORMAT_VERSION = 0x81 }</p>

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.

16.7 PollForPlatformEventMessage command

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

This command is implemented to poll for events on synchronous transports and shall be the only method for retrieving large event messages from a PLDM terminus. This command provides a multiple part transfer mechanism to retrieve event messages, which have variable data fields. Large messages are broken into chunks of data, the size of which shall be negotiated through the EventMessageBufferSize command. An example of such a message is the pldmPDRRepositoryChgEvent.

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

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

1674 If an event is sent in asynchronous mode and the terminus is switched to polling mode before the Event
1675 Receiver acknowledges the event, then the terminus shall send the oldest event on the next polling
1676 request unless the terminus overwrites the event.

1677 The formatVersion field shall be fixed at 0x01 for this specification.

1678 Figure 22 shows an example flow that demonstrates switching to polled event transfer to receive an event
1679 with large event data. When the Event Receiver gets a pldmMessagePollEvent, this is a signal that an
1680 event with a large amount of event data is next to be transferred. The Event Receiver then uses the
1681 PollForPlatformEventMessage command with TransferOperationFlag set to GetFirstPart to initiate the
1682 transfer and the dataTransferHandle provided in the pldmMessagePollEvent. In response, the PLDM
1683 terminus supplies the first chunk of data along with a transfer handle for the next portion and a
1684 transferFlag of Start, which indicates that this is the first chunk and there is at least one more. The Event
1685 Receiver then retrieves the next chunk in the same fashion, using the nextDataTransferHandle supplied
1686 in the previous response. So long as the response message transferFlag field is set to Middle, the Event
1687 Receiver knows that more data is waiting to be retrieved and repeats this process using the most recently
1688 received nextDataTransferHandle to obtain the next data chunk each time. Finally, when the transferFlag
1689 comes back as End, the Event Receiver knows the transfer is complete and can verify the
1690 eventDataIntegrityChecksum against the reassembled event data. The eventID from the
1691 pldmMessagePollEvent should match the eventID returned from the PollForPlatformEventMessage
1692 command. Assuming the transfer was successful, the Event Receiver can now acknowledge receipt of
1693 the event and switch back to asynchronous transfer of events by sending a final
1694 PollForPlatformEventMessage command with TransferOperationFlag set to AcknowledgementOnly.

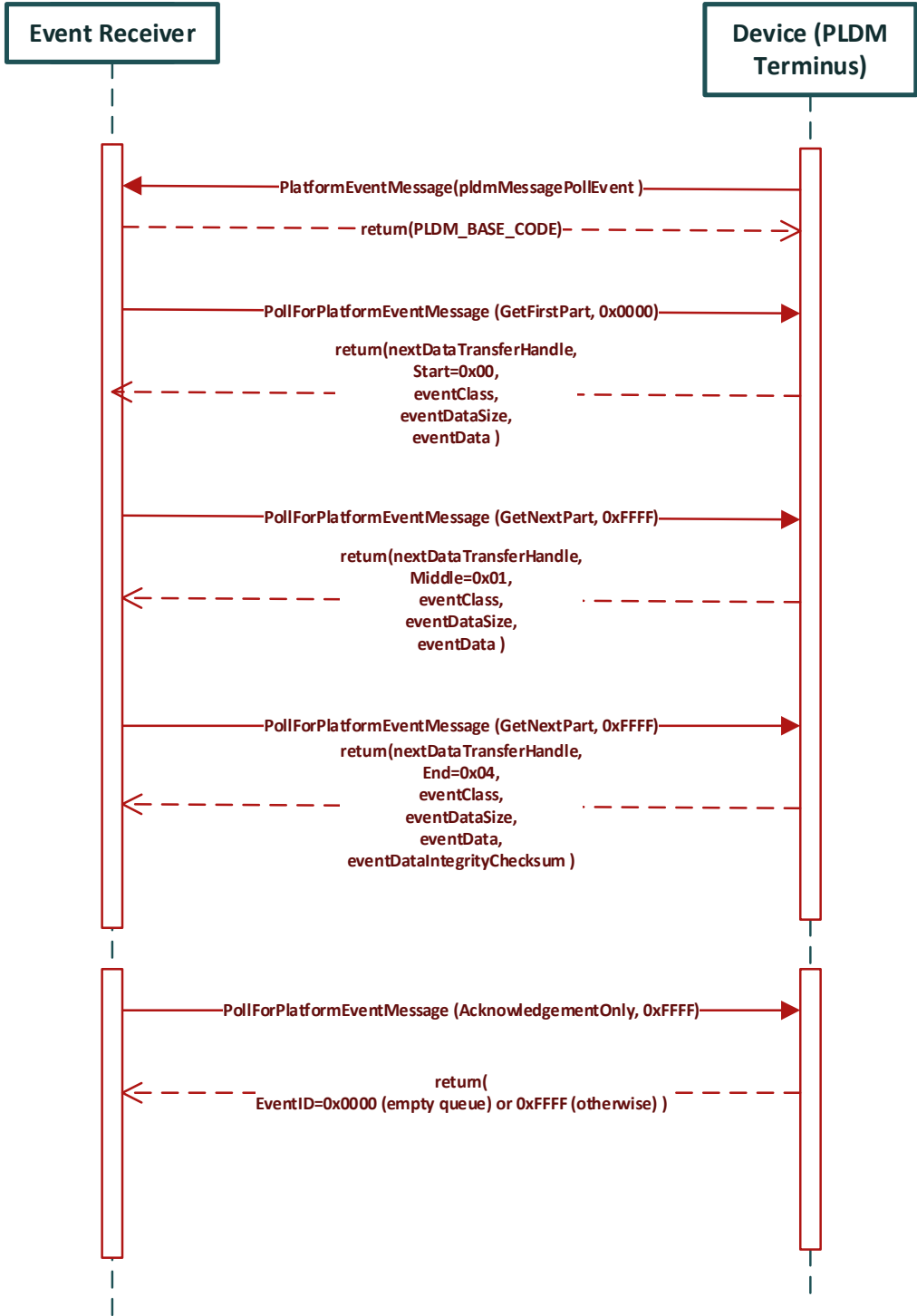


Figure 22: Switching from asynchronous eventing to poll for an event with large data

1697

Table 16 – PollForPlatformEventMessage command format

Type	Request data
uint8	formatVersion Version of the event format (the format and definition of the following bytes): 0x01 for this specification.
enum8	TransferOperationFlag The operation flag that indicates whether this is the start of the transfer. Possible values: {GetNextPart=0x00, GetFirstPart=0x01, AcknowledgementOnly=0x02}
uint32	dataTransferHandle A handle that is used to identify a package data transfer. This handle is ignored by the responder when the TransferOperationFlag is set to AcknowledgementOnly. If the PollForPlatformEventMessage command is executed because of a eventData format for pldmMessagePollEvent, this will be the dataTransferHandle value returned from the pldmMessagePollEvent.
uint16	eventIDToAcknowledge An event previously received that should be acknowledged; The event receiver shall use the null value 0x0000 when requesting the first entry from the terminus' event queue or the GetFirstPart of a multipart event as indicated by an eventData format for pldmMessagePollEvent. The event receiver shall use the special value 0xFFFF when in the middle of a multipart event transfer (TransferOperationFlag is GetNextPart)
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_DATA_TRANSFER_HANDLE -= 0x80, UNSUPPORTED_EVENT_FORMAT_VERSION = 0x81, EVENT_ID_NOT_VALID=0x82 }
uint8	TID Terminus ID for the terminus from which event messages are being supplied
uint16	eventID 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.
uint32	nextDataTransferHandle A handle that is used to identify the next portion of the transfer. This field shall be omitted if eventID is 0x0000 or 0xFFFF.
enum8	TransferFlag 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.
uint8	eventClass 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.

uint32	eventDataSize The size in bytes of the eventData field below. (Does not include eventDataIntegrityChecksum.) This field shall be omitted if eventID is 0x0000 or 0xFFFF.
Type	Response data (continued)
var	eventData A chunk of Event data, based on the eventClass, in a buffer sized as negotiated in the EventMessageBufferSize command. This field shall be omitted if eventID is 0x0000 or 0xFFFF.
uint32	eventDataIntegrityChecksum 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). 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). 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.

1698

1699 16.8 EventMessageSupported Command

1700 The EventMessageSupported command is optional for this specification version. It is recommended,
1701 however, that a terminus supports this command if the terminus accepts the SetEventReceiver command.
1702 This command returns a list of eventClass supported by the terminus. The enumeration values for the
1703 eventClass are defined in Table 11.

1704

Table 17 – EventMessageSupported command format

Type	Request data
uint8	formatVersion Version of the event format (the format and definition of the following bytes): 0x01 for this specification version
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, UNSUPPORTED_EVENT_FORMAT_VERSION = 0x81 }

enum8	synchronyConfiguration This value indicates the messaging style most recently configured via the SetEventReceiver command: value: { NOT_CONFIGURED = 0x00, // SetEventReceiver command not received // or eventMessageGlobalEnable is set to // disable ASYNCHRONOUS_MESSAGING = 0x01, // Asynchronous messaging SYNCHRONOUS_MESSAGING = 0x02 // Poll-based messaging ASYNCHRONOUS_WITH_HEARTBEAT = 0x03 // Asynchronous messaging, heartbeat }
Type	Response data (continued)
bitfield8	synchronyConfigurationSupported 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. [7:4] - Reserved for future use [3] - Asynchronous messaging with heartbeat [2] - Synchronous (poll-based) messaging [1] - Asynchronous messaging, no heartbeat [0] - Reserved; shall be 0b.
uint8	numberEventClassReturned The count N of eventClass enumerated bytes returned in this response
uint8	eventClass [0] The first eventClass message the device can generate. The eventClass values are defined in Table 11.
uint8	eventClass [1] The second eventClass message the device can generate. The eventClass values are defined in Table 11.
uint8	...
uint8	eventClass [N-1] The last eventClass message the device can generate. The eventClass values are defined in Table 11.

1705

1706 16.9 EventMessageBufferSize Command

1707 The EventMessageBufferSize command is optional for this specification version. It is recommended,
1708 however, a terminus supports this command if the terminus accepts the SetEventReceiver command.
1709 This command communicates the maximum size of the event receiver buffer that can hold a single event
1710 message. The response is the maximum size of the terminus buffer that can transmit a single event
1711 message. The smaller of the two values shall be the negotiated event message size. Any event message
1712 that exceeds the negotiated event message buffer size shall be retrieved by the event receiver using the

1713 PollForPlatformEventMessage command. The terminus shall send the pldmMessagePollEvent to the
 1714 PLDM event receiver when an event message exceeds the negotiated buffer size.

1715 In the event that this command is not invoked, a default message buffer size of 256 bytes shall be in
 1716 effect.

1717 **Table 18 – EventMessageBufferSize command format**

Type	Request data
uint16	eventReceiverMaxBufferSize This is the maximum buffer to hold an event message transferred from the terminus to the event receiver. This value represents the size of the PLDM header and PLDM payload; medium specific header information shall not be included in this calculation.
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. This value represents the size of the PLDM header and PLDM payload; medium specific header information shall not be included in this calculation. The smaller of eventReceiverMaxBufferSize and terminusMaxBufferSize shall be the negotiated size for all event messages regardless of asynchronous or polled.

1718 16.10 eventData format for sensorEvent

1719 Table 19 defines the format of the eventData field in PLDM Event Messages for the sensorEvent class.
 1720 This field includes event data for PLDM state sensor and numeric sensor events, and for events related to
 1721 changes of the sensor's operational state.

1722 **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	previousEventState The event state value for the state from which the present event state was entered. See Table 41 for the definition of eventState. 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.
<i>For sensorEventClass = numericSensorState</i>	
enum8	eventState The eventState value from the state change that triggered the event message. See Table 30 for the enumeration values of eventState.
enum8	previousEventState The eventState value for the state from which the present state was entered. See Table 30 for the enumeration values of eventState. 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).
enum8	sensorDataSize The bit width and format of reading and threshold values that the sensor returns value: { uint8, sint8, uint16, sint16, uint32, sint32 }
uint8 sint8 uint 16 sint16 sint32 uint32	presentReading The present value indicated by the sensor. The sensorDataSize field returns an enumeration that indicates the number of bits used to return the value.
<i>For sensorEventClass = sensorOpState</i>	
enum8	presentOpState The sensorOperationalState value from the state change that triggered the event message. See Table 30 for the enumeration values of sensorOperationalState.
enum8	previousOpState The sensorOperationalState value for the state from which the present state was entered. See Table 30 for the enumeration values of sensorOperationalState. 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.

1723 16.11 eventData format for effectorEvent

1724 Table 20 defines the format of the eventData field in PLDM Event Messages for the effectorEvent class.
 1725 This field supports events for changes of the effector's operational state.

1726

Table 20 – effectorEvent class eventData format

Type	Request data
uint16	effectorID 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.
enum8	effectorEventClass value: { effectorOpState // Events from a PLDM state or numeric effector that are related to // changes of the effector's operational state }
<i>For effectorEventClass = effectorOpState</i>	
enum8	presentOpState The effectorOperationalState value from the state change that triggered the event message.
enum8	previousOpState The effectorOperationalState value for the state from which the present state was entered. 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.

1727 16.12 eventData format for redfishTaskExecutedEvent

1728 Table 21 defines the format of the eventData field in PLDM Event Messages for the redfishTaskExecuted
 1729 class. This field supports PLDM events for completion of a long-running Redfish Task as defined in
 1730 [DSP0218](#).

1731

Table 21 – redfishTaskExecutedEvent class eventData format

Type	Request data
uint32	resourceID 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
uint16	operationID Operation associated with the Task that has completed execution

1732

1733 16.13 eventData format for redfishMessageEvent

1734 Table 22 defines the format of the eventData field in PLDM Event Messages for the redfishMessageEvent
 1735 class. A PLDM event may contain one or more Redfish Events. See [DSP0218](#) for information on how
 1736 PLDM for Redfish Device Enablement uses RDE events and [DSP0266](#) for information on the events
 1737 themselves.

1738 Redfish Events contain timestamps. For RDE Devices that do not contain realtime clocks, the timestamp
 1739 shall be set to a sentinel value of zero. When decoding Redfish Events with the timestamp set to the zero
 1740 sentinel, the MC may substitute a current timestamp.

1741

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 cause the event to 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 .

16.14 eventData format for pldmPDRRepositoryChgEvent

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

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

1. PDR records to be removed (deleted) from the event receiver’s repository shall be first, grouped either in a single event message or as individual event messages.
2. PDR records to be added to the event receiver’s repository shall be after the deleted records, grouped either in a single event message or as individual event messages.

1757 3. The existing PDR records to be modified in the event receiver's repository shall be last, grouped
1758 either in a single event message or as individual event messages.

1759 For example, if a hard drive is added to a storage enclosure under control of an intelligent storage
1760 adapter, the terminus could indicate the addition of PDRs representing the newly added hard drive in one
1761 event message followed by another event message indicating the affected Entity Association PDRs. The
1762 event receiver, which may also be the primary repository manager, only needs to retrieve the affected
1763 PDRs rather than the entire repository.

1764 Another example is if an entire storage enclosure is removed, the number of affected PDRs returned in
1765 this event message may exceed the MCTP baseline transmission unit size. In this example, setting the
1766 eventDataFormat to a special value of "refreshEntireRepository" is the best choice.

1767 The goal of this event is to avoid retrieving the entire device PDR repository for a small device PDR
1768 repository differences.

1769

1770 **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 [$N_R - 1$]

1771

1772

Table 24 – pldmPDRRepositoryChgEvent changeRecord format

Type	Request data
enum8	eventDataOperation { refreshAllRecords, recordsDeleted, recordsAdded, recordsModified } 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”. 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.
uint8	numberOfChangeEntries The number of change entries N_E following this field.
uint32	changeEntry [0] 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. 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. Alternatively, the terminus may provide a list of PDR Record Handles, which the MC can use as input to the GetPDR command.
uint32	changeEntry [1]
...	...
uint32	changeEntry [$N_E - 1$]

1773

16.15 eventData format for pldmMessagePollEvent

1775 Table 25 defines the format of the eventData field in PLDM Message Poll Event. This event typically
 1776 signals the event receiver that a polling command is needed to retrieve a large event message from the
 1777 terminus.

1778

Table 25 – pldmMessagePollEvent class eventData format

Type	Request data
uint8	formatVersion Version of the event format (the format and definition of the following bytes): 0x01 for this specification.
uint16	eventID Identifier for the event that requires multipart transfer. This value shall be between 0x0001 and 0xFFFF as the values 0x0000 and 0xFFFF are reserved for PollForPlatformEventMessage command.
uint32	dataTransferHandle A handle that is used to identify the event data to be received via the PollForPlatformEventMessage command. The PLDM Terminus is permitted to set a NULL (zero) value to maintain a strict FIFO within its internal event queue. If the PLDM Terminus sets a NULL for this field, the PLDM Event Receiver shall poll all the events currently queued in the PLDM Terminus.

16.16 eventData format for heartbeatTimerElapsedEvent

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

Table 26 – heartbeatTimerElapsedEvent class eventData format

Type	Request data
uint8	formatVersion Version of the event format (the format and definition of the following bytes): 0x01 for this specification.
uint8	sequenceNumber 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.

17 PLDM Numeric Sensors

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

17.1 Sensor readings, data sizes

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

17.2 Units and reading conversion

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

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

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

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

17.3 Reading-only or threshold-based numeric sensors

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

17.4 Readable and settable thresholds

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

17.5 Update/polling intervals and states updates

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

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

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

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

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

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

17.6 Thresholds, Present State, and Event State

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

17.6.1 Threshold severity levels

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

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.

17.6.2 Upper and lower thresholds

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

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

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

17.6.3 Present State

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

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

17.6.4 Event State

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

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

17.7 Manual re-arm and auto re-arm sensors

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

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

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

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

17.8 Event message generation

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

17.9 Threshold values and hysteresis

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

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

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

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

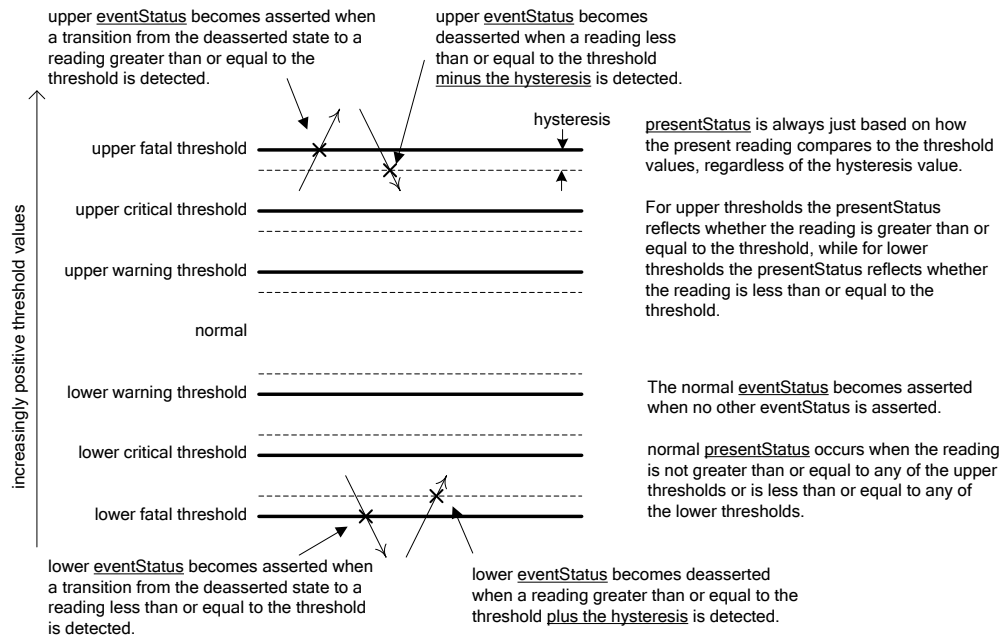


Figure 23 – Numeric sensor threshold and hysteresis relationships

18 PLDM Numeric Sensor commands

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

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

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.

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

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

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

18.1 SetNumericSensorEnable command

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

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	sensorEventMessageEnable 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.
Type	Response data
enum8	completionCode 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. }

18.2 GetSensorReading command

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

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

Table 30 – GetSensorReading command format

Type	Request data
uint16	sensorID A handle that is used to identify and access the sensor special values: 0x0000, 0xFFFF reserved
bool8	rearmEventState true = manually re-arm EventState after responding to this request Re-arming causes the sensor to enter the “initializing” state until it updates its presentState and eventState. 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. 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. false = no manual re-arm

Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_SENSOR_ID = 0x80, REARM_UNAVAILABLE_IN_PRESENT_STATE = 0x81 }
enum8	sensorDataSize The bit width and format of reading and threshold values that the sensor returns value: { uint8, sint8, uint16, sint16, uint32, sint32 }
enum8	sensorOperationalState The state of the sensor itself value: { enabled, disabled, unavailable, statusUnknown, failed, initializing, shuttingDown, inTest } <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	sensorEventMessageEnable value: { noEventGeneration, eventsDisabled, eventsEnabled, opEventsOnlyEnabled, stateEventsOnlyEnabled }

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	presentReading The present value indicated by the sensor 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.
--	--

18.3 GetSensorThresholds command

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

Table 31 describes the format of the command.

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

18.4 SetSensorThresholds command

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

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

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

Table 32 describes the format of the command.

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 }

18.5 RestoreSensorThresholds command

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

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 }

18.6 GetSensorHysteresis command

The GetSensorHysteresis command is used to read 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. Table 34 describes the format of the command.

1989

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 }
For sensorDataSize = uint8 or sint8	
uint8 sint8	hysteresis value
For sensorDataSize = uint16 or sint16	
uint16 sint16	hysteresis value
For sensorDataSize = uint32 or sint32	
uint32 sint32	hysteresis value

1990

18.7 SetSensorHysteresis command

1991
1992
1993
1994

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.

1995

Table 35 – SetSensorHysteresis 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 following hysteresis value that is being set into 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 GetSensorHysteresis command.
<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	completionCode value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80 }

1996

18.8 InitNumericSensor command

1997 The InitNumericSensor command is typically used by the Initialization Agent function (see clause 15) to
1998 initialize PLDM Numeric Sensors. The command may also be used as an interface for “virtual sensors,”
1999 which do not actually poll and update their own state but instead rely on another management controller
2000 or system software to set their state.

2001 Implementations should avoid virtual sensors that require initialization by the Initialization Agent function.
2002 Conflicts could occur if the sensor needs to be accessed by the Initialization Agent function at the same
2003 time it is being accessed as a virtual sensor. Typically, however, a virtual sensor would not require
2004 initialization by the Initialization Agent function.

2005 Table 36 describes the format of the command.

2006

Table 36 – InitNumericSensor 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 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. This parameter is applied to the sensor <i>after</i> all other fields (sensorPresentState, eventMsgEnable, and numericReadingSetting) have been applied to the sensor. value: { enabled, disabled, unavailable }
enum8	sensorPresentState The expected present state of the numeric sensor. See the description of the presentState field in Table 30.
enum8	eventMsgEnable This value is used to enable or disable event message generation from the sensor. value: { enableEventMessages, disableEventMessages, noChange=0xFF // Do not alter the present event enable setting. }
bool8	setNumericReading value: { false, true } True directs the receiver to accept the following numericReadingSetting.
var	numericReadingSetting 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.
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80 }

2007

19 PLDM State Sensors

2008

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](#).

2009

2010

2011

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.

2012

2013

2014 This specification also includes support for the definition of vendor-specific state sets using the OEM
2015 State Set PDR. (See 28.10 for more information.)

2016 If a state sensor is reporting events or status and is based on a numeric sensor, the state sensor shall
2017 use the threshold and hysteresis values for the associated numeric sensor for state change notification.
2018 State Sensors that reflect logical states, such as redundancy, are device dependent and these sensor
2019 types are outside the scope of this specification.

2020 20 PLDM State Sensor commands

2021 This clause describes the commands for accessing PLDM State Sensors per this specification. The
2022 command numbers for the PLDM messages are given in clause 30.

2023 If PLDM State Sensors are implemented, the Mandatory/Conditional (M/C) requirements shown in Table
2024 37 apply.

2025 **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.

2026 ^[1] Required for sensors that are to be initialized through the Initialization Agent function.

2027 20.1 SetStateSensorEnables command

2028 The SetStateSensorEnables command is used to set enable or disable sensor operation and event
2029 message generation for sensors within a PLDM Composite State Sensor. Event message generation is
2030 optional for a sensor. Table 38 describes the format of the command.

2031 **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 }

2032

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.

2033

20.2 GetStateSensorReadings command

2034

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

2035

2036

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

2037

2038

2039

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

2040

2041

Table 40 describes the format of the command.

2042

Table 40 – GetStateSensorReadings command format

Type	Request data
uint16	sensorID A handle that is used to identify and access the simple or composite sensor special values: 0x00, 0xFFFF = reserved
bitfield8	sensorRearm 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. For each bit position [n] from n = 0 to compositeSensorCount-1, the bit setting operates as follows: 0b = do not re-arm sensor [n]+1 1b = re-arm sensor [n]+1
uint8	reserved value: 0x00
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES, INVALID_SENSOR_ID=0x80 }
unit8	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
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 and event state for a particular set of sensor information contained within the state sensor. The stateField structure is defined in Table 41.

2043

Table 41 – GetStateSensorReadings stateField format

Type	Description
enum8	sensorOperationalState The state of the sensor itself See Table 30 for the enumeration values of sensorOperationalState.
enum8	presentState 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.

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>

2044 20.3 InitStateSensor command

2045 The InitStateSensor command is typically used by the Initialization Agent function (see clause 15) to
 2046 initialize PLDM State Sensors. The command may also be used as an interface for virtual sensors, which
 2047 do not actually poll and update their own state but instead rely on another management controller or
 2048 system software to set their state.

2049 Implementations should avoid virtual sensors that require initialization by the Initialization Agent function.
 2050 Conflicts could occur if the sensor needs to be accessed by the Initialization Agent function at same time
 2051 it is being accessed as a virtual sensor. Typically, however, a virtual sensor would not require initialization
 2052 by the Initialization Agent function.

2053 Table 42 describes the format of the command.

2054 **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	completionCode 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 }

2055

Table 43 – InitStateSensor initField format

Type	Description
enum8	sensorOperationalState 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. This parameter is applied to the sensor after all other fields (sensorPresentState and eventMsgEnable) have been applied to the sensor. value: { enabled, disabled, unavailable }
enum8	sensorPresentState 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. value: { dependent on sensor State Set }
enum8	eventMsgEnable This value is used to enable or disable event message generation from the sensor. value: { enableEvents, disableEvents, noChange=0xFF } noChange means do not alter the present setting.

2056 21 PLDM effecters

2057 PLDM effecters provide a general mechanism for controlling or configuring a state or numeric setting of
2058 an entity. PLDM effecters are similar to PLDM sensors, except that entity state and numeric setting values
2059 are written into an effector rather than read from it.

2060 PLDM commands are specified for writing the state or numeric setting to an effector. Effecters are
2061 identified by and accessed using an EffectorID that is unique for each effector within a given terminus.
2062 Corresponding PDRs provide basic semantic information for effecters, such as what type of states or
2063 numeric units the effector accepts, what terminus and EffectorID value are used to access the effector,
2064 which entity the effector is associated with, and so on.

2065 21.1 PLDM State Effecters

2066 PLDM State Effecters provide a regular command structure for setting state information in order to
2067 change the state of an entity. Effecters use the same PLDM State Sets definitions as PLDM State
2068 Sensors, but instead of using the state set information to interpret the value that is read from a sensor,
2069 the state sets are used to define the value to write to an effector. Like PLDM Composite State Sensors,
2070 PLDM State Effecters can be implemented and accessed as composite state effecters where a single

2071 EffectorID is used to access a set of state effecters. This enables multiple states to be set using a single
 2072 command and to share a single PDR that provides the basic information for the effecters.

2073 21.2 PLDM Numeric Effecters

2074 PLDM Numeric Effecters provide a regular command structure for setting a numeric value for a
 2075 controllable parameter of an entity. Numeric effecters use the same definition of units as the units for
 2076 readings returned by numeric sensors (see 27.2). For example, a numeric effector could be used to set a
 2077 value for revolutions per second.

2078 21.3 Effector semantics

2079 An effector has a meaning or use that is associated with what an effector does or is used for. This will be
 2080 referred to as the "effector semantic", or just the "semantic."

2081 Although PLDM effecters provide a straightforward mechanism for setting a state or numeric value for an
 2082 entity, conveying the semantic of how that state or numeric value affects the entity, or how the setting
 2083 should be used, is not always straightforward.

2084 Suppose a numeric effector is defined for setting a fan speed. A PDR for the numeric effector can readily
 2085 indicate that the effector is for "Physical Fan 1", and that "Fan 1" is contained by Processor 1. The PDR
 2086 can also indicate that the units for the setting are "RPM". However, this does not convey what the RPM is
 2087 actually doing. For example, is the RPM a speed limit or a target speed?

2088 Additionally, other information may be necessary for understanding how the effector is to be used. If a fan
 2089 speed needs to be set because one or more temperatures have become too high, how does the user of
 2090 PLDM know which temperatures are associated with the fan, and what RPM value should be set for a
 2091 particular temperature?

2092 The information required to describe the meaning and use of an effector can vary significantly depending
 2093 on how generic or specific the use is to the platform implementation. The level of generality of effector
 2094 semantics in PLDM is categorized as shown in Table 44.

2095 **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.

2096 The most generic and efficient use of effectors comes when they fall into the state sets or units only
 2097 category and use standard state set or units definitions. The second most generic and efficient use of
 2098 effectors is when they use a standard defined Semantic ID. Thus, if new standard effector semantics
 2099 need to be defined, it should be first examined whether a new state set or units definition should be
 2100 added to the specifications, or whether a new Semantic ID should be added.

2101 21.4 PLDM and OEM effector semantic IDs

2102 Effector Semantic ID values are specified in [DSP0249](#). A range of values is reserved for definition by the
 2103 DMTF PLDM specifications and another range of values is available for OEM (vendor-defined) effector
 2104 semantics. When the OEM range is used, the semantic is identified and optionally named using an OEM
 2105 Effector Semantic PDR. The use of the OEM Effector Semantic PDR is similar to how OEM units, entities,
 2106 and state sets are defined within the PDRs.

2107 22 PLDM effector commands

2108 This clause describes the commands for accessing PLDM effectors per this specification. The command
 2109 numbers for the PLDM messages are given in clause 30.

2110 If PLDM Numeric Effectors or PLDM State Effectors are implemented, the Mandatory (M) requirements
 2111 shown in Table 45 apply.

2112 **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.

2113 ^[1] Required if one of more numeric effectors are implemented

2114 ^[2] Required if one or more state effectors are implemented

2115 22.1 SetNumericEffectorEnable command

2116 The SetNumericEffectorEnable command is used to enable or disable effector operation. A disabled
 2117 effector cannot have its state updated. An effector may have a default state that it automatically returns to
 2118 when it is disabled. An effector may also be able to be returned to its default state through the

2119 SetStateNumericEffectorValue command. The PLDM Numeric Effector PDR can describe a numeric
2120 effector and whether it has a default state.

2121 NOTE The Numeric Effector PDR effectorID type, in clause 28.11 Numeric Effector PDR has been changed in
2122 version 1.1.1 of this specification from uint8 to uint16 to be consistent with SetNumericEffectorEnable command.

2123 Table 46 describes the format of this command.

2124 **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 }

2125 22.2 SetNumericEffectorValue command

2126 The SetNumericEffectorValue command is used to set the value for a PLDM Numeric Effector. Table 47
2127 describes the format of this command.

2128 **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, }

2129 **22.3 GetNumericEffectorValue command**

2130 The GetNumericEffectorValue command is used to return the present numeric setting of a PLDM Numeric
2131 Effector. Table 48 describes the format of this command.

2132 **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>

2133 22.4 SetStateEffectorEnables command

2134 The SetStateEffectorEnables command is used to enable or disable effector operation. A disabled
 2135 effector cannot have its state updated. An effector may have a default state that it automatically returns to
 2136 when it is disabled. An effector may also be able to be returned to its default state through the

2137 SetStateEffectorStates command. The PLDM State Effector PDR describes a state effector and whether
 2138 it has a default state. Table 49 describes the format of this command.

2139 **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 }

2140 **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.

22.5 SetStateEffectorStates command

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

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. }

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 }

22.6 GetStateEffectorStates command

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

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.

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 }

23 PLDM Event Log commands

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

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

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

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

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

23.1 GetPLDMEventLogInfo command

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

Table 56 – GetPLDMEventLogInfo command format

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

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	mostRecentEraseTimestampUTCOffset The UTC offset for the log entry timestamp in increments of 1/2 hour. special value: 0xFF = unspecified
uint40	mostRecentEraseTimestampSeconds 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.
uint8	mostRecentEraseTimestamp100s 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.

23.2 EnablePLDMEventLogging command

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

Table 57 – EnablePLDMEventLogging command format

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

23.3 ClearPLDMEventLog command

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

2181

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). }

2182 **23.4 GetPLDMEventLogTimestamp command**

2183 The GetPLDMEventLogTimestamp command returns a snapshot of the present PLDM Event Log
 2184 Timestamp time. Table 59 describes the format of this command.

2185 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.

2186 **23.5 SetPLDMEventLogTimestamp command**

2187 The SetPLDMEventLogTimestamp command can be used to set the PLDM Event Log Timestamp time.

2188 Some implementations may not implement the ability to set the timestamp to 1/100 of a second resolution
 2189 and will round the time up or down to match the resolution that it supports. Therefore, the timestamp

2190 value in the response may vary from what was submitted because of rounding. The returned value may
 2191 also vary due to delays in command response processing within the terminus.

2192 Implementations are required to support a 1 second or finer resolution for the timestamp. Table 60
 2193 describes the format of this command.

2194 **Table 60 – SetPLDMEventLogTimestamp command format**

Type	Request data
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 This value is ignored if the implementation only timestamps entries to a one-second resolution.
enum8	logUpdateEvent value: { 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. }

2195

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)

2196 23.6 ReadPLDMEventLog command

2197 The ReadPLDMEventLog command can be used iteratively to read all or part of the entries in the PLDM
2198 Event Log. Entries are returned one at a time. The data for one or more entries may be requested. Table
2199 61 describes the format of this command.

2200 To use the command to start reading from the first entry in the log:

- 2201 • Set entryID to 0 and transferOperationFlag to GetFirstPart.
- 2202 • Issue the command to get the first portion of data for the first entry in the log.
- 2203 • Take the nextEntryID and nextTransferOperationFlag data from the response and use it as the
2204 entryID and transferOperationFlag for the next request.
- 2205 • Repeat this until the desired number of entries have been read or the end of the log has been
2206 reached.

2207 The FindPLDMEventLogEntry command can be used to get the entryID for an entry that is at an offset
2208 into the log, or that has a timestamp that is older or newer than a given value. This entryID can then be
2209 used in the ReadPLDMEventLog command, along with setting transferOperationFlag = GetFirstPart, to
2210 begin reading the log starting with the found entry.

2211

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,</p> <p>INVALID_TRANSFER_OPERATION_FLAG=0x81,</p> <p>INVALID_ENTRY_ID=0x82,</p> <p>}</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	splitEntry value: { full, // All of the data for the entry is provided in the entryData field. firstFragment, // The eventData for the entry is split across ReadPLDMEventLog messages. // The entryData field holds the first portion of the data for the entry. middleFragment, // The eventData for the entry is split across ReadPLDMEventLog messages. // The entryData field holds a middle portion of the data for the entry. lastFragment // The eventData for the entry is split across ReadPLDMEventLog messages. // The entryData field holds the last portion of the data for the entry. }
–	PLDMEventLogData The data or partial data for the requested PLDM Event Log entry. Entries are transferred starting from the oldest to the newest.
<i>If splitEntry = lastFragment</i>	
uint8	transferCRC 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/I ² C 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.

2212

Table 62 – PLDMEventLogData format

Type	Field
uint8	transferredDataSize If splitEntry = full, then dataSize = number of bytes of entryData for the entire entry. If splitEntry = firstFragment, middleFragment, or lastFragment, then dataSize = number of bytes of entryData for the portion that is being transferred.
–	transferredEntryData 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.

2213 23.7 GetPLDMEventLogPolicyInfo command

2214 The GetPLDMEventLogPolicyInfo command returns details about the different log clearing policies that
2215 are supported for the particular PLDM Event Log implementation. Table 63 describes the format of this
2216 command.

2217

2218

Table 63 – GetPLDMEventLogPolicyInfo command format

Type	Request data
enum8	logClearingPolicy 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. value: { fillAndStop, FIFO, clearOnAge }
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
bitfield8	configurableParameterSupport This information and the following fields are specific to the logClearingPolicy that was selected in the request. [7:5] – reserved [4:3] – 00b = M and MPercentage are not configurable. 01b = M is configurable 10b = MPercentage is configurable. 11b = reserved [2:1] – 00b = N and NPercentage are not configurable. 01b = N is configurable. 10b = NPercentage is configurable. 11b = reserved [0] – 1b = Age is configurable.
uint32	NMin The smallest number that the implementation accepts or uses as a value for N for the given logClearingPolicy (see 13.4). special value: Return 0x00000000 if the policy implementation uses NPercentage instead of N, or if the policy does not use an N value.
uint32	NMax The largest number that the implementation accepts or uses as a value for N for the given logClearingPolicy (see 13.4). special value: Return 0x00000000 if the policy implementation uses NPercentage instead of N, or if the policy does not use an N value.
Type	Response data (continued)
uint8	NPercentageMin The smallest number that the implementation accepts or uses as a value for NPercentage for the given logClearingPolicy (see 13.4). value: 1 to 100; all other values = reserved special value: Return 0x00 if the policy implementation uses N instead of NPercentage, or if the policy does not use an NPercentage value.

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>

2219 23.8 SetPLDMEventLogPolicy command

2220 The SetPLDMEventLogPolicy command is used to select and configure the PLDM Event Log clearing
 2221 policies. Table 64 describes the format of the command.

2222

Table 64 – SetPLDMEventLogPolicy command format

Type	Request data
enum8	selectedLogClearingPolicy 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. value: { fillAndStop, FIFO, clearOnAge }
enum8	setOperation value: { 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. 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. 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. }
uint32	N The number of entries that will be automatically cleared for the given selectedLogClearingPolicy. See 13.4 for a description of the log clearing policies. 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.
uint8	NPercentage 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. value: 1 to 100; all other values = reserved 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.
uint32	M 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. 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.

Type	Request data (continued)
uint8	MPercentage <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	age <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	completionCode <p>value: { PLDM_BASE_CODES }</p>

2223 23.9 FindPLDMEventLogEntry command

2224 This command can be used to obtain the Entry ID value for the first entry in the Event Log that meets the
 2225 identified search parameter. This value can then be used in the ReadPLDMEventLog command to start
 2226 reading the log from that entry onward. The search parameters support finding the first entry that is newer
 2227 or older than a specified timestamp value, or the entry that corresponds to a particular offset from the
 2228 start or the present end of the log. Table 65 describes the format of this command.

2229 NOTE The order of fields in the response message for this command has been changed to having the
 2230 completionCode before the entryID in version 1.2.0 of this specification; this achieves consistency with all other
 2231 PLDM commands.

2232

Table 65 – FindPLDMEventLogEntry command format

Type	Request data
enum8	searchType value: {newerThan, olderThan, offsetFromStart, offsetFromEnd}
uint32	startingPoint The EntryID for the log entry or the offset from which searching will start. Searches include the entry at the identified starting point. The search always occurs in the direction from the start of the log (first entries) to the end of the log (last entries). If searchType = newerThan or olderThan: 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. If searchType = offsetFromStart: The value identifies the Nth entry from the start of the log. For example, if starting point = 10 the search will start with the 10 th entry at the beginning of the log. An error completionCode shall be returned if the value exceeds the number of entries in the log. If searchType = offsetFromEnd: 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 91 st entry. An error completionCode shall be returned if the value exceeds the number of entries in the log.
compareTimestamp <i>The compareTimestamp fields are only present when searchType = newerThan or olderThan.</i> <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> <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>	
sint8	compareTimestampUTCOffset The UTC offset for the log entry timestamp in increments of 1/2 hour. special value: 0xFF = unspecified
uint40	compareTimestampSeconds 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.
uint8	compareTimestamp100s 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.
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.
--------	---

24 PLDM State Sets

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

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

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

25 Platform Descriptor Records (PDRs)

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

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

25.1 PDR Repository updates

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

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

25.2 Internal storage and organization of PDRs

The PLDM specifications do not place any requirements on how PDRs are internally stored or organized within the device or devices that implement the PDR Repository. PDRs may be compressed, stored with additional pointers, sorted, cross indexed, split, replicated, and so on, as long as the information meets

the byte order and formats specified for the PDR commands. The byte order and formats for PDRs are specified in tables for the different PDR types in clause 28.

25.3 PDR types

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

25.4 PDR record handles

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

25.5 Accessing PDRs

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

26 PDR Repository commands

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

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

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

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

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

26.1 GetPDRRepositoryInfo command

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

2301 See 25.5 for more information about accessing PDRs. Table 67 describes the format of this command.

2302 **Table 67 – GetPDRRepositoryInfo command format**

Type	Request data
–	none
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }
enum8	repositoryState 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. }
timestamp104	updateTime 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".
timestamp104	OEMUpdateTime 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.
uint32	recordCount Total number of PDRs in this repository
uint32	repositorySize Size of the PDR Repository in bytes. This value provides information that can be used for helping estimate buffer size requirements when accessing PDRs. 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. An implementation is allowed to round this number up to the nearest kilobyte (1024 bytes).
uint32	largestRecordSize 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. An implementation is allowed to round this number of up to the nearest 64-byte increment.
uint8	dataTransferHandleTimeout The minimum interval, in seconds, that a dataTransferHandle value remains valid after it was delivered in the response of a GetPDR or FindPDR command. 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. }

26.2 GetPDR command

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

26.2.1 GetPDR command format

Table 68 describes the format of the GetPDR command.

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. The nextRecordHandle shall only be used in subsequent GetPDR operations when the transferOperationFlag is set to GetFirstPart. For multipart transfers, the current recordHandle shall be the same for all the transfers of the current multipart transfer operation. 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. When this field is equal to zero and the transferOperationFlag is set to GetFirstPart, this indicates this is a single part PDR transfer. When this field is not equal to zero, the recordHandle value sent when the transferOperationFlag is set to GetFirstPart is used on subsequent portion requests as the nextRecordHandle until the nextDataTransferHandle is returned as zero, indicating the multipart transfer for the specific PDR is complete. special value: { returns 0x0000_0000 if there is no remaining data. }
Type	Response data (continued)
enum8	transferFlag Indicates what portion of the PDR is being transferred value: {Start = 0x00, Middle = 0x01, End = 0x04, StartAndEnd = 0x05}
uint16	responseCount The number of recordData bytes returned in this response special value: { returns 0x0000 if the requestCount was 0x0000 }
(var)	recordData PDR data bytes. This field is absent if responseCount = 0x0000. The number of PDR data bytes returned in this field must match responseCount.
<i>If transferFlag = End</i>	
uint8	transferCRC 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/I ² C 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.

2310 26.2.2 Single-part and multipart transfers

2311 The data from a given PDR may be accessed using a single-part or multipart transfer. A single transfer
2312 occurs when the entire PDR content is delivered using a single GetPDR command response. A multipart
2313 transfer is required either when the record data exceeds the amount of data that the responder can return
2314 using a single response, or when it exceeds the amount of data that the requester can accept in a single
2315 response. In this case, the GetPDR command is used iteratively to retrieve the first portion of the record
2316 and then subsequent portions. Additional information and requirements for multipart transfers is provided
2317 in 26.2.7.

2318 Partial transfers from the beginning of a record are allowed. That is, a requester is not required to read
2319 out an entire record if only the beginning portion of the record data is of interest.

2320 26.2.3 PDR recordHandle

2321 The recordHandle is an opaque value that is used by the implementation of the PDR Repository to
2322 identify individual records. This value is obtained from the response data of a previous instance of the
2323 GetPDR command. A special value of 0x0000_0000 is used to retrieve the first PDR in the repository.

2324 The recordHandle remains the same during a multipart transfer until all portions of the specific PDR have
2325 been retrieved.

2326 Some implementations may use the recordHandle as a direct offset into storage memory, others may use
2327 it as offset that is relative to the start of the PDR data, and others may use it as a table or list index.

2328 **26.2.4 PDR recordHandle retention**

2329 The recordHandle values that are used to access a particular PDR may change when the
2330 recordChangeNumber is changed. recordHandle values are also not guaranteed to endure across
2331 connections to the given PLDM terminus that is implementing the command. A party that needs to re-
2332 establish a connection to the terminus must assume that any PDR recordHandle values that it previously
2333 had are no longer valid. If any multipart transfers were not completed before the connection was re-
2334 established, those transfers must be restarted from the beginning.

2335 **26.2.5 PDR recordChangeNumber**

2336 The recordChangeNumber provides a mechanism for preventing the use of invalid PDR data if a record's
2337 data gets updated while the record was in the process of being read out. The mechanism helps ensure
2338 that a requester does not get the first parts from an earlier version of the record and remaining parts from
2339 a later version of the record. The recordChangeNumber can also be used to help a requester scan and
2340 identify which PDRs may have changed after an update to the PDR Repository has occurred.

2341 To accomplish this, the PDR recordChangeNumber that is returned in the GetPDR response is required
2342 to change whenever the data of a PDR changes during a multipart access of the PDR. The party that is
2343 accessing a PDR gets the recordChangeNumber when the first part of the record is returned. This
2344 number is then used as one of the input parameters when retrieving the remaining parts of the record.

2345 The PLDM responder compares this number against the present recordChangeNumber that is associated
2346 with the record. If there is a mismatch, the PLDM responder returns an error completionCode. The
2347 requester can then handle the error by starting the PDR transfer over.

2348 It is recommended that an implementation update the recordChangeNumber only for records that have
2349 changed due to an update. However, implementations may elect to update the recordChangeNumber for
2350 some or all unchanged records. This latter approach can be used for small and simple implementations in
2351 which PDR exits and updates are rare, but should be avoided in large implementations in which the party
2352 that is accessing the PDR data may see significant delays due to the unnecessary re-reading and
2353 handling of PDRs that have not actually changed.

2354 **26.2.6 PDR Repository timestamp and PDR Repository locking**

2355 The recordChangeNumber mechanism protects against inconsistent data only on a per record basis; it
2356 does not automatically protect against inconsistencies that may occur due to individual updates of
2357 interrelated records. For example, if record A and B are interrelated and both need synchronized updates,
2358 it is possible that a party could access the records at a time when A has been updated but B has not. The
2359 individual records would be correct, but their interrelationship could be incorrect.

2360 The party that is updating the PDRs can lock the repository while updates are occurring (the mechanisms
2361 used for updating and locking the PDRs are outside this specification). In this case, commands such as
2362 the GetPDR command will return an error completionCode indicating that the repository records are
2363 inaccessible because an update is in progress. Update-in-progress status is also available in the
2364 GetPDRRepositoryInfo command.

2365 A party that updates records in a PDR Repository while PLDM command handling is active must either:
2366 lock the PDRs and update the timestamp and recordChangeNumber values before making the repository
2367 available; or update the timestamp and recordChangeNumber values as each individual updated record
2368 is made available through PLDM.

2369 The PDR Repository has a timestamp that can be read using the GetPDRRepositoryInfo command. The
2370 timestamp value is updated whenever changes are made to the repository. A party that is accessing

multiple PDRs and relying on an interrelationship between those records should check the timestamp value after retrieving the records to verify that a repository update did not occur while the records were being accessed.

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

26.2.7 Multipart PDR transfers

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

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

The recordHandle of the GetFirstPart of the multipart PDR transfer operation shall be used until the nextDataTransferHandle value equals zero, indicating the current PDR data multipart transfer operation is complete.

26.2.8 PDR dataTransferHandle retention

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

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

26.2.9 Multipart PDR transfer termination and timeouts

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

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

26.2.10 Reuse of prior request values

Except for the first part of a PDR, an implementation is not required to support returning a previously transferred portion of a PDR after the transfer has progressed to a later portion. For example, if the first three portions of a PDR have been transferred, the implementation may not allow a re-transfer of the second portion without restarting the transfer from the beginning. If an implementation does accept

2413 request parameters that were used for reading an earlier portion of a given PDR, it must return the same
2414 PDR data that was returned for the original request.

2415 **26.3 FindPDR command**

2416 The FindPDR command is provided to improve the efficiency of common types of access to a Primary
2417 PDR Repository. The FindPDR command is primarily designed to provide operations that can assist a
2418 MAP in using information from the PDRs to instantiate CIM objects and associations.

2419 The FindPDR command returns the PLDMHandleType and PLDMHandle values for a particular PDR or
2420 set of PDRs, depending on the parameters that were passed in the request. The response can also
2421 include the first portion of the PDR data. The response from the FindPDR command can then be used
2422 with the GetPDR command to read the PDR or the remaining portions of the PDR.

2423 To reduce implementation and validation complexity, the FindPDR command does not provide a generic
2424 search engine but supports only a limited number of different preconfigured queries that are restricted to
2425 using particular key fields within the PDRs.

2426 For example, the FindPDR command can be used to find all the PDRs that have a particular
2427 PLDMTerminusHandle, or Entity Association PDRs that have a common Container ID. It can also be used
2428 to find Numeric Sensor PDRs that share a particular type of monitored numeric unit, such as temperature,
2429 or state sensors that use a particular state set. However, the FindPDR command does not support less
2430 common operations such as finding records that have a particular hysteresis value setting or state
2431 sensors that implement a particular state from within a state set.

2432 The findParameters field holds the PDRTYPE-specific search fields. The format of findParameters is
2433 identified by the parameterFormatNumber that is passed in the request. The findParameters value may
2434 be applicable to more than one PDRTYPE. The parameterFormatNumber and PDRTYPE field in the
2435 request are used together to identify which PDRs should be searched. Table 70 lists the values for
2436 parameterFormatNumber and the PDRTYPE values that are associated with each
2437 parameterFormatNumber. Table 71 lists the different PDR fields that make up the findParameters value
2438 for each different parameterFormatNumber.

2439 If the PDRTYPE field value is set to 0, all of the PDRTYPE values that are specified for the
2440 parameterFormatNumber in Table 70 are searched. Otherwise, only PDRs that have the given PDRTYPE
2441 value are searched.

2442 For example, if PDRTYPE = 0 and parameterFormatNumber = 7, all PDRs with PDRTYPE values that are
2443 identified for searching with parameterFormatNumber = 7 are searched: Numeric Effector Initialization,
2444 State Effector Initialization, and Effector Auxiliary Names. If the PDRTYPE is set to the value for State
2445 Effector Initialization PDR, only State Effector Initialization PDRs are searched.

2446 The findParameters value is included in each request to eliminate the need for implementations to retain
2447 the findParameters value when a multi-PDR find operation is being done.

2448 Table 69 describes the format of this command.

2449

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)	recordData PDR data bytes. This field is absent if responseCount = 0x0000. Otherwise, the number of PDR data bytes returned in this field must match responseCount.
-------	---

2450

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	

2451 ^[1] The entire contents of the repository can be read by using this format along with PDRType = ANY and PLDMTerminusHandle set
2452 for "wildcard."

2453 ^[2] The PLDMTerminusHandle parameter must be set for "wildcard" when using this format to search for Event Log PDRs.

2454 ^[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
2455 any of the indicated "OEM" PDRType values and match a particular vendorIANA.

2456

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														•	

26.4 RunInitAgent command

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

Table 72 – RunInitAgent command format

Type	Request data
bitfield8	initConditionEmulation 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. value: { 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. 0x01 = PLDMSubsystemPowerUp, // Directs the Initialization Agent to take the same steps // as it would when the PLDM subsystem becomes // powered up. 0x02 = SystemHardReset, // Directs the Initialization Agent to take the same steps // as it would following a system hard reset. 0x03 = SystemWarmReset, // Directs the Initialization Agent to take the same steps // as it would following a system warm reset. 0x04 = PLDMTerminusOnline // Directs the Initialization Agent to initialize the // terminus that has a TID that matches the TID // parameter in this request. }
uint8	TID Terminus ID for the terminus to be initialized when the initConditionEmulation field in this request is set to PLDMTerminusOnline. special value: The value in this field is ignored when the initConditionEmulation field in this request is set to any value other than PLDMTerminusOnline.
Type	Response data
enum8	completionCode value: { PLDM_BASE_CODES }

26.5 GetPDRRepositorySignature command

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

2471 management controller is expected to compare the current value to the previous value to detect a
 2472 terminus PDR Repository modification.

2473 **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.

2474 **27 PDR definitions**

2475 This clause describes certain important characteristic parameters that are provided within the PDRs for
 2476 interpreting the readings and settings of sensors and effecters.

2477 **27.1 Sensor types**

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

- 2479 • The PLDM Numeric Sensor is used to obtain a numeric value for a monitored parameter. The
 2480 sensor definition also optionally includes returning state information based on whether the
 2481 numeric reading has crossed one or more defined threshold levels.
- 2482 • The PLDM State Sensor/PLDM Composite State Sensor is used to obtain the present state of a
 2483 monitored parameter. The PLDM sensor access commands allow an implementation to provide
 2484 multiple sets of state information using a single access command. When this is done, the
 2485 implementation is referred to as providing a Composite State Sensor.

2486 **27.2 Effector types**

2487 PLDM contains two basic types of effecters that are described using PDRs:

- 2488 • The PLDM Numeric Effector is used to set a numeric value for a monitored parameter.
- 2489 • The PLDM State Effector/PLDM Composite State Effector is used to set the present state of a
 2490 monitored parameter. The PLDM effector access commands allow an implementation to provide
 2491 multiple sets of state information using a single access command. When this is done, the
 2492 implementation is referred to as providing a Composite State Effector.

2493 **27.3 State sets**

2494 State information is returned using an enumeration called a “state set.” Each state set has a different ID
 2495 number. This number is used within the PDRs to identify what particular state set a sensor or effector is
 2496 using. See clause 24 for more information.

2497 27.4 Sensor and effector units

2498 This subclause and following subclauses describe the fields that are used within PDRs to define and
2499 describe sensor and effector units and related characteristics such as accuracy, tolerance, and resolution.

2500 The type of units that are associated with the value that a sensor returns or monitors, or that an effector
2501 controls, such as volts or amps, is identified in the PDRs by a sensorUnits enumeration, listed in Table
2502 74. Unless otherwise indicated, the units apply to all numeric properties of the sensor, such as the sensor
2503 reading, threshold values, and resolution.

2504 Vendor-defined units are identified by a special value for OEMUnit. A special PDR called the OEM Unit
2505 PDR is used to define the meaning of the OEMUnit when it is used in the PDRs that describe a sensor or
2506 effector. Refer to 28.9 for more information about how OEMUnits are used in PDRs.

2507

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

2508 **27.4.1 Base units**

2509 The base unit of measurement that is associated with the reading values returned by a PLDM Numeric
 2510 Sensor or set into a PLDM Numeric Effector is represented by the combination of three fields from the

2511 PDR for the sensor: baseUnits, unitModifier, and rateUnits. These fields are interpreted according to the
2512 following formula:

2513 **Sensor/Effecter Units = baseUnit * 10^{unitModifier} rateUnit**

2514 For example, if baseUnits is Volts and the unitModifier is -6, the units of the values returned are
2515 microvolts.

2516 If the rateUnits property is set to a value other than None, the units are further qualified as rate units. In
2517 the preceding example, if rateUnits is set to Per Second, the values returned by the sensor are in
2518 microvolts/second.

2519 **27.4.2 Auxiliary units**

2520 In some cases, additional modification of the base unit of the sensor might be required. For example,
2521 acceleration is commonly given in units such as "meters per second per second". The PDRs include a
2522 provision for modifying the base units with an additional set of units called auxiliary units. Auxiliary units
2523 are defined by three elements: auxUnit, auxUnitModifier, and auxRateUnit. These elements are used in
2524 combination with the base units as follows:

2525 **Sensor/Effecter Units = baseUnit * 10^{unitModifier} [rel] auxUnit * 10^{auxUnitModifier} rateUnit auxRateUnit**

2526 [rel] is the relationship between the base unit and the auxiliary unit, as follows:

2527 rel = enum8 { dividedBy, multipliedBy }

2528 And:

2529 dividedBy implies a "/" or "per" relationship, such as "per foot"

2530 multipliedBy implies a "*" operation, such as "foot*lbs (foot-lbs)"

2531 auxUnit and auxRateUnit shall not be used if an equivalent definition can be made using only base units.

2532 **27.4.3 Units for use with CIM**

2533 Developers are cautioned that PLDM units may include types of units that are not presently supported by
2534 standard CIM objects such as CIM_Sensor. PLDM supports additional types of units because certain
2535 types of sensors or effecters may be used within a platform management subsystem but are not exposed
2536 through CIM, or are mapped into CIM using proprietary CIM extensions. Parties developing platform
2537 management subsystems in which sensors are intended to be exposed as CIM objects should first verify
2538 which types of sensors and units are supported by CIM and the CIM profiles.

2539 **27.4.4 OEM (vendor-defined) sensor units**

2540 OEM (vendor-defined) sensor units are identified in PLDM sensor PDRs when the OEMUnit value from
2541 Table 74 is used for the baseUnit or auxUnit. The semantic information of an OEMUnit can then be
2542 further described using an OEM Sensor Units PDR that is associated with the particular sensor that is
2543 returning the OEMUnit. Multiple OEM Sensor Units PDRs can be defined if there is a need for defining
2544 more than one type of OEM unit. Additionally, multiple PLDM Sensor PDRs can be associated with a
2545 particular OEM Sensor Units PDR.

2546 **27.5 Counters**

2547 A counter is a numeric sensor that returns a value that returns a count. PLDM does not define any
2548 requirements on whether a counter must increment, decrement, or both, or whether it does so
2549 sequentially or monotonically, and so on.

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

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

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

27.6 Accuracy, tolerance, resolution, and offset

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

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

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

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

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

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

27.6.1 Additional information about numeric sensor/effector tolerance

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

27.6.2 Examples of accuracy, tolerance, and resolution use

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

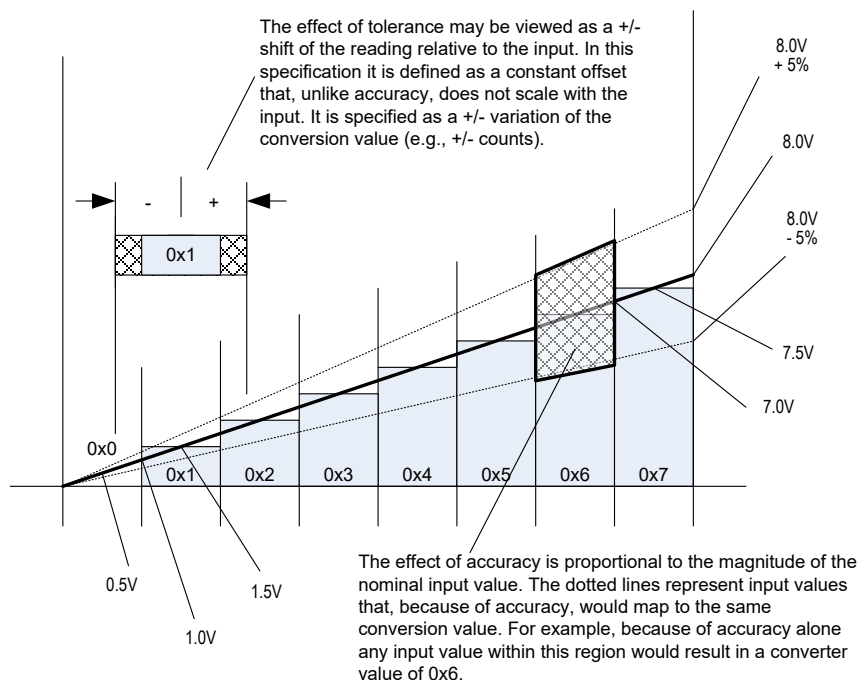


Figure 24 – Accuracy, tolerance, and resolution example

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

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

0x0	→ 0 V to 1.0 V (0.5 V ± 0.5 V)
0x1	→ 1.0 V to 2.0 V (1.5 V ± 0.5 V)
0x2	→ 2.0 V to 3.0 V (2.5 V ± 0.5 V)
0x3	→ 3.0 V to 4.0 V (3.5 V ± 0.5 V)
0x4	→ 4.0 V to 5.0 V (4.5 V ± 0.5 V)
0x5	→ 5.0 V to 6.0 V (5.5 V ± 0.5 V)
0x6	→ 6.0 V to 7.0 V (6.5 V ± 0.5 V)
0x7	→ 7.0 V to 8.0 V (7.5 V ± 0.5 V)

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

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

2614 Note that this follows the Cartesian coordinate formula for a line: $y = Mx + B$

2615 Now, suppose that the implementation could add a negative D.C. offset of 0.5 V to the input. Then the
2616 center point for a reading of 0.0 V would correspond to 0.0 V, and a reading of 0x0 would correspond to a
2617 range of 0.0 V \pm 0.5 V instead of 0.0 V to 1.0 V. In this case, the conversion would then be $V = (\text{resolution}$
2618 $\times \text{reading}) + 0.0 \text{ V}$. There is now no offset relative to the center of the reading value because of a D.C.
2619 offset. If the converted negative offset of 4.0 V was connected to the input, a reading of 0x0 would now
2620 correspond to -3.5 V \pm 0.5 V and a reading of 111b would correspond to 3.5 V \pm 0.5 V.

2621 It is very common for an A/D converter implementation to have a D.C. offset that needs to be accounted
2622 for when converting a reading to the corresponding nominal input value. The party that implements the
2623 hardware for the sensor needs to provide this offset value as well at the resolution (step size per count)
2624 so that the basic conversion of the reading can be accomplished.

2625 After the basic conversion of the reading is done, the effects of accuracy and tolerance may need to be
2626 taken into account. For example, if someone is depending on the reading to determine whether
2627 something has failed, it is important to understand how much error might be in the reading so that a
2628 failure is not falsely assessed for a healthy component.

2629 For PLDM, the effects of accuracy and tolerance are considered to be orthogonal to one another and
2630 additive. First consider the effect of accuracy. Suppose the accuracy of the sensor is specified as $\pm 5\%$.
2631 Using that figure, a value of 001b will nominally correspond to 1.5 V \pm 5%, but because of quantization
2632 and accuracy, any value from 1.0 V \pm 5% to 2.0 V \pm 5% (a range of 0.95 V to 2.10 V) could result in a
2633 reading of 0x1.

2634 The next step is to factor in tolerance. The quantization within a converter is never perfect; some slight
2635 variation always exists in the comparison points that yield a particular converter output. Instead of the
2636 conversion ranges being evenly spaced as shown in Figure 24, some ranges may be a little wider and
2637 others a little narrower. The effect of this is that in an actual implementation, borderline values such as
2638 1.99 V or 2.01 V, for example, may sometimes yield a value of 0x1 and sometimes 0x2.

2639 Tolerance in PLDM is defined as an error in the quantization that is applied to all counts of the converter
2640 equally. Because PLDM sensors are all specified as returning integer values, any errors in the reading
2641 will always result in an integral number of counts. Thus, tolerance is specified as a +/- effect on the count.

2642 The tolerance value is typically used to account for quantization errors in A/D conversion circuitry that
2643 occur because of effects such as D.C. voltage offsets within the circuit. For example, suppose the input to
2644 an A/D converter that monitors voltage was shifted up by a constant amount, as would be the case if a
2645 D.C. offset was added to the input. Per the figure, if a D.C. offset error of 0.25 V were added when
2646 converting, the input reading 0x01 would represent a range that actually goes from 0.75 V to 1.75 V
2647 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
2648 cause a reading of 0x1 to be returned instead 0x0. Thus, because of this offset error, the reading would
2649 be one count higher than it was intended to be for inputs in that range. Similarly, with the same offset, a
2650 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
2651 2.00 V would also result in a reading that is one count higher than intended.

2652 This does not mean that all conversions are off by one count. In this example, the reading is incorrect
2653 only for inputs that are in the range caused by the offset. A reading of 0x1 would be correctly returned for
2654 an input of 1.5 V. The reading can thus be incorrect by 0 counts or +1 counts depending on what range
2655 the input value is in. In this case, the tolerance would be specified as +1/-0 counts.

2656 Manufacturing variations and tolerances in A/D conversion circuitry mean that both positive and negative
2657 offsets are possible. This is why it is typical to see a specification of ± 1 count for tolerance. In many
2658 implementations, tolerance is specified as ± 1 count for these types of conversions. Because resolution is
2659 given in units of 1 count, tolerance and resolution may sometimes appear to equate to the same value.
2660 However, tolerance and resolution should not be misinterpreted as being the same thing.

2661 Lastly, in some cases PLDM Numeric Sensors will return values such as counts or other measurements
2662 that to not use a conversion process that can introduce errors in the reading. In this case, the tolerance is
2663 specified as ± 0 counts.

2664 **27.6.3 Accuracy, tolerance, and resolution relationship to thresholds**

2665 Accuracy, tolerance, and resolution must all be taken into account to generate a threshold that does not
2666 generate a "false positive" (a false indication of a failure). For example, if accuracy, tolerance, and
2667 resolution are not taken into account when calculating the threshold for a warning level, it is possible that
2668 an input could be assessed as being within the warning range when the input was actually near the limit
2669 of the normal range.

2670 A consequence of avoiding false positives is that for a particular range a value that is actually within the
2671 intended warning range can be assessed as being within the normal range. That is, false positives are
2672 avoided at the cost of having the possibility of 'false negatives'. However, in most implementations it is
2673 considered better to avoid the false alarms that false positives would cause. Whether to design thresholds
2674 to avoid false positives or false negatives is a choice of the system implementation.

2675 Because it is the more common case, the following examples describe how thresholds may be calculated
2676 to avoid false positives.

2677 EXAMPLE: An 8-bit A/D converter monitoring a 5.0 V nominal signal where the sensor has been designed such
2678 that the 5.0 V level corresponds to a reading of C0h and the 0.0 V level corresponds to a reading of
2679 00h (as shown by Figure 25A). Assume the converter implementation has a specified worst-case
2680 accuracy of $\pm 4\%$, and a tolerance of ± 1 count.

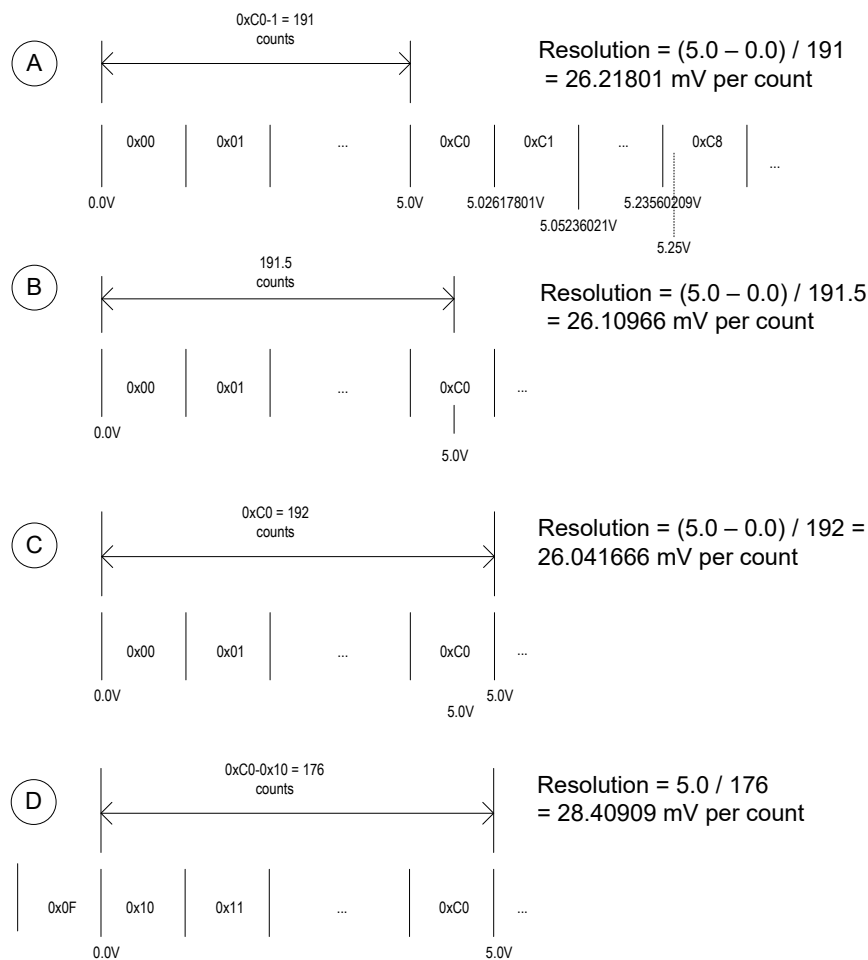


Figure 25 – Figuring resolution from the design

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

Resolution

= 5.0 V / (C0h -1) = 26.17801 mV

Accuracy

= ± 4% (given, from the design)

Tolerance

= ± 1 count (given) = ± 26.17801 mV

Now, suppose it is necessary to calculate an upper critical threshold for the 5.0 V + 5% point (5.25 V) where this threshold will not produce "false positives" (falsely return 'critical') across the range of accuracy, tolerance, and resolution. The following example shows steps that can be used to calculate a threshold suitable for a PLDM Numeric Sensor:

2694 Step 1: Divide the target threshold value by the resolution to find how many counts correspond to
 2695 5.25 V:

2696 $5.25 \text{ V} / 26.17801 \text{ mV} = 200.55 \text{ counts}$
 2697 (which puts the 5.25 V point within the nominal range of reading 0xC8, as shown in
 2698 Figure 25A)

2699 Step 2: Factor in the tolerance:

2700 **Important:** Because tolerance is specified as an error, a "+" count for tolerance means that
 2701 the reading may be higher than it should be, and a "-" count means that the reading may be
 2702 lower than it should be. To account for these errors, the "-" tolerance value should be added
 2703 to upper thresholds, and the "+" tolerance value subtracted from lower thresholds. This is
 2704 particularly important when the plus and minus tolerance values are different from one
 2705 another.

2706 $200.55 + 1 = 201.55 \text{ counts}$

2707 Step 3: Account for the effect of accuracy:

2708 $201.55 * 1.04 = 209.612 \text{ counts}$

2709 Step 4: Round up (because an A/D converter cannot give a non-integer count)

2710 $209.612 \rightarrow 210 \text{ counts} = 0xD2$

2711 This yields a threshold value of 210, which corresponds to 5.497 V. This shows that even though a
 2712 threshold of 5.25 V is being targeted, it is necessary to set the threshold to a value that, because of the
 2713 effects of accuracy, tolerance, and resolution, could allow the actual monitored value to be as high as
 2714 5.497 V in some implementations before a threshold match would be detected.

2715 The calculations for lower thresholds are the same, except that negative values for the accuracy,
 2716 tolerance, and resolution are used.

2717 Figure 25 illustrates what to be aware of when deriving the values for resolution from an implementation.
 2718 To get an accurate value for resolution, it is important to know whether the input values that correspond to
 2719 a particular reading are given as values that are at the point of change (quantization point) between
 2720 successive readings, are a nominal "center point" of a reading, or a combination of the two. (The
 2721 difference in the resolution value between Figure 25A and Figure 25C is almost 0.5%. This shows that a
 2722 nontrivial amount of error could be introduced if the implementer uses the wrong calculation point for its
 2723 implementation).

2724 Lastly, area D in Figure 25 shows that offsets in the implementation also need to be taken into account.
 2725 Offset adds a new first step to the threshold calculation:

2726 Step 0: Take the target threshold and subtract (or add, depending on the implementation) the D.C.
 2727 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.

27.8 Numeric effector conversion formula

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

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

Where:

X = integer setting value for the effector

Y = target setting in Units

m = resolution from PDR in Units

B = offset from PDR in Units

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

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

28 Platform Descriptor Record (PDR) formats

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

28.1 Common PDR header format

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

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

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

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	recordChangeNumber See 26.2.3 for more information.
uint16	dataLength The total number of PDR data bytes following this field.

28.2 PDR type values

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

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.
-----	---------	------------

28.3 Terminus Locator PDR

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

Type	Description
uint8	terminusInstance This field is used to differentiate between different PLDM termini if the device contains more than one PLDM terminus.
UUID	deviceUID 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. A PLDM terminus that is identified using this type of ID must support the GetTerminusUID command.
<i>terminusLocatorValue for terminusLocatorType = MCTP_EID:</i>	
uint8	EID A MCTP EID that is assigned to an MCTP Endpoint that provides the transport protocol termination for a PLDM terminus
<i>terminusLocatorValue for terminusLocatorType = SMBusRelative</i>	
UUID	UID 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> ".
uint8	busNumber A bus number for the bus to which the device is connected, relative to the controller that owns the bus. 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.
uint8	slaveAddress The SMBus or I ² C slave address for the device that is providing the [7:1] - SMBus or I ² C slave address value. [0] - 0b.
<i>terminusLocatorValue for terminusLocatorType = systemSoftware</i>	
enum8	softwareClass { unspecified, other, systemFirmware, OSloader, OS, CIMprovider, otherProvider, virtualMachineManager }

Type	Description
UUID	UUID 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.

28.4 Numeric Sensor PDR

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

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

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.

Type	Description
enum8	sensorInit Indicates whether the sensor requires initialization by the initializationAgent. value: { noInit, // 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
enum8	baseUnit The base unit of the reading returned by this sensor. See 27.4 for more information. value: { see Table 74 }
sint8	unitModifier A power-of-10 multiplier for the baseUnit. See 27.4 for more information.
enum8	rateUnit value: { None, Per MicroSecond, Per MilliSecond, Per Second, Per Minute, Per Hour, Per Day, Per Week, Per Month, Per Year }
uint8	baseOEMUnitHandle 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.
enum8	auxUnit The base unit of the reading returned by this sensor. See 27.4 for more information. value: { see Table 74 }
sint8	auxUnitModifier A power-of-10 multiplier for the auxUnit. See 27.4 for more information.
enum8	auxrateUnit value: { None, Per MicroSecond, Per MilliSecond, Per Second, Per Minute, Per Hour, Per Day, Per Week, Per Month, Per Year }
enum8	rel The relationship between the base unit and the auxiliary unit, as follows: value = { dividedBy, multipliedBy} dividedBy implies a "/" or "per" relationship, such as "per foot" multipliedBy implies a "*" operation, such as "foot*lbs (foot-lbs)"

Type	Description
uint8	auxOEMUnitHandle This value is used to locate the PLDM OEM Unit PDR that defines the OEMUnit if the OEMUnit value is used for the auxUnit.
bool8	isLinear Indicates whether a sensor is linear or dynamic in its range. For example, this value can be used by a MAP to populate the IsLinear attribute of CIM_NumericSensor. value: This field is set to "true" to show that a sensor is linear.
enum8	sensorDataSize The bit width and format of reading and threshold values that the sensor returns value: { uint8, sint8, uint16, sint16, uint32, sint32 }
real32	resolution The resolution of the sensor in Units (see 27.7).
real32	offset A constant value that is added in as part of the conversion process of converting a raw sensor reading to Units (see 27.7).
uint16	accuracy Given as a +/- percentage in 1/100ths of a % from 0.00 to 100.00. For example, the integer value 510 corresponds to $\pm 5.10\%$. See 27.6 for more information.
uint8	plusTolerance 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. See 27.6 for more information about how tolerance is defined and used.
uint8	minusTolerance 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. See 27.6 for more information about how tolerance is defined and used.
uint8 sint8 uint16 sint16 uint32 sint32	hysteresis 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. The size of this field is identified by sensorDataSize. value: 1 or greater special value: 0 = sensor does not use hysteresis

Type	Description
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>
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>

Type	Description
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>
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>

Type	Description
uint8 sint8 uint16 sint16 uint32 sint32 real32	normalMax 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.
uint8 sint8 uint16 sint16 uint32 sint32 real32	normalMin 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.
uint8 sint8 uint16 sint16 uint32 sint32 real32	warningHigh 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.
uint8 sint8 uint16 sint16 uint32 sint32 real32	warningLow 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.
uint8 sint8 uint16 sint16 uint32 sint32 real32	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.
uint8 sint8 uint16 sint16 uint32 sint32 real32	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.
uint8 sint8 uint16 sint16 uint32 sint32 real32	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.

Type	Description
uint8 sint8 uint16 sint16 uint32 sint32 real32	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.

28.5 Numeric Sensor Initialization PDR

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

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 }

Type	Description
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 }
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.

2806 28.6 State Sensor PDR

2807 The State Sensor PDR provides the sensorID for a composite state sensor within a PLDM terminus and
2808 the number of sensors, and the state set and the possible state values for each sensor that is accessed
2809 through the given sensorID. The record also identifies the entity that is being monitored by the sensor.
2810 Only one set of fields exists for the entity identification information. Therefore, all sensors in this record
2811 must be associated with the same entity. Table 80 describes the format of this PDR.

2812

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.
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 State Sensor Initialization PDR // that should be used to initialize the sensor. enableSensor, // When the Initialization Agent runs, it enables this sensor using // a SetStateSensorEnables command to set the // operationalState. disableSensor. // When the Initialization Agent runs, it disables this sensor using // the SetStateSensorEnables command. } }
bool8	sensorAuxiliaryNamesPDR true = sensor has a Sensor Auxiliary Names PDR false = sensor does not have an associated Sensor Auxiliary Names PDR
uint8	compositeSensorCount The number of state sensors in the terminus that are accessed under the sensorID given in this PDR value: 0x01 to 0x08
var	possibleStates One instance of State Sensor Possible States Fields (see Table 81) for each sensor in the PLDM State Sensor, up to sensorCount.

2813

Table 81 – State Sensor possible states fields format

Type	Description
uint16	stateSetID A numeric value that identifies the PLDM State Set that is used with this sensor
uint8	possibleStatesSize The number of bytes (M) in the following possibleStates bitfield value: 0x01 to 0x20 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.
bitfield8 x M	possibleStates [subset of the State Set that is supported] 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. 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. 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. [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. ... [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. [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. [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.

2814 28.7 State Sensor Initialization PDR

2815 The State Sensor Initialization PDR contains values that direct the Initialization Agent's initialization of a
2816 particular PLDM Single or Composite State Sensor. This action includes enabling or disabling PLDM
2817 Event Message generation for individual sensors within the PLDM Composite State Sensor and directing
2818 whether a particular sensor will assess an event if the initialization state value does not match the present
2819 state of the sensor.

2820 The PDR always has eight state values (stateValue0 through stateValue7). Dummy values must be used
2821 (0x00 is recommended) if the implementation does not have a sensor that corresponds to a particular
2822 offset. Table 82 describes the format of the PDR.

2823

Table 82 – State 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
bitfield8	initConditions Identifies under which conditions the Initialization Agent must initialize or reinitialize these sensors 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. [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 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. special value: {0xFF = do not set the sensorOperationalStates}
bitfield8	sensorInitMask Identifies which sensors within the composite state sensor require initialization [7] – 1b = state sensor at offset 7 requires initialization 0b = state sensor at offset 7 does not require initialization [6] – 1b = state sensor at offset 6 requires initialization 0b = state sensor at offset 6 does not require initialization ... [2] – 1b = state sensor at offset 2 requires initialization 0b = state sensor at offset 2 does not require initialization [1] – 1b = state sensor at offset 1 requires initialization 0b = state sensor at offset 1 does not require initialization [0] – 1b = state sensor at offset 0 requires initialization 0b = state sensor at offset 0 does not require initialization

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.

28.8 Sensor Auxiliary Names PDR

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

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

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	sensorCount [1..M] <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	nameStringCount <p>Number of following pairs [0..N] of nameLanguageTag + sensorName fields for sensor[1].</p>
strASCII	nameLanguageTag [1] <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	sensorName [1] <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]

28.9 OEM Unit PDR

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

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

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

Table 84 – OEM Unit PDR format

Type	Description
–	commonHeader <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]

28.10 OEM State Set PDR

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

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

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

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

- 2860 2) The TID, sensorID, sensorOffset, and state values (that is, eventState and previousEventState)
2861 are read from the message.
- 2862 3) The TID is used to look up the Terminus Locator Record and obtain the PLDMTerminusHandle
2863 value that is associated with the TID.
- 2864 4) PLDMTerminusHandle and sensorID values are used to look up the PLDM State Sensor PDR
2865 for the sensor.
- 2866 5) The Sensor Offset is used to get the stateSetID from the PLDM State Sensor PDR. If the
2867 stateSetID is in the range of standard IDs, the meaning of the state value is given according to
2868 the stateSetID defined by the state set identified in [DSP0249](#).
- 2869 6) Otherwise the stateSetID from the PLDM State Sensor PDR is used as an
2870 OEMStateSetIDHandle to look up the OEM State Set PDR that defines the OEM State Set. The
2871 PDR identifies the OEM that defined the state set and provides the OEM-specified State Set
2872 number (OEMStateSetID) for the state set. The state value from the event message can be
2873 used to locate the OEM State Value Record in the PLDM OEM State Set PDR that provides a
2874 name string for the particular OEM-defined state.

2875 Table 85 describes the format of the PDR.

2876 **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.

2877

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]

2878 28.11 Numeric Effector PDR

2879 The Numeric Effector PDR is used to describe the semantics of a PLDM Numeric Effector to a party such
 2880 as a MAP. It also includes the factors that are used for converting raw sensor readings to normalized
 2881 units. The PDR also identifies the entity on which the effector is operating. Table 87 describes the format
 2882 of the PDR.

2883 NOTE The Numeric Effector PDR effectorID type in this clause has been changed in version 1.1.1 of this
 2884 specification from uint8 to uint16 to be consistent with SetNumericEffectorEnable command.

2885

2886

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

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>

2887 **28.12 Numeric Effector Initialization PDR**

2888 The Numeric Effector Initialization PDR reports the values that are used when a PLDM Effector Sensor is
 2889 initialized by a PLDM Initialization Agent. Table 88 describes the format of this PDR.

2890 **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.

28.13 State Effector PDR

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

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

2895

Table 90 – State Effector Possible States fields format

Type	Description
uint16	stateSetID A numeric value that identifies the PLDM State Set that is used with this effector.
uint8	possibleStatesSize The number of bytes (M) in the possibleStates bitfield. value: 0x01 to 0x20 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.
bitfield8 x M	possibleStates [subset of the State Set that is supported] 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. 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. 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. [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 ... [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 [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 [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

2896 28.14 State Effector Initialization PDR

2897 The State Effector Initialization PDR describes settings that the Initialization Agent uses to initialize a
 2898 PLDM Single or Composite State Effector.

2899 The PDR always has eight state values. Dummy values must be used (0x00 is recommended) if the
 2900 implementation does not have an effector that corresponds to a particular offset. Table 91 describes the
 2901 format of the PDR.

2902 **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	effectorInitMask Identifies which effecters within the composite state effector require initialization <ul style="list-style-type: none"> [7] – 1b = state effector at offset 7 requires initialization 0b = state effector at offset 7 does not require initialization [6] – 1b = state effector at offset 6 requires initialization 0b = state effector at offset 6 does not require initialization ... [2] – 1b = state effector at offset 2 requires initialization 0b = state effector at offset 2 does not require initialization [1] – 1b = state effector at offset 1 requires initialization 0b = state effector at offset 1 does not require initialization [0] – 1b = state effector at offset 0 requires initialization 0b = state effector at offset 0 does not require initialization
bitfield8	effectorOpStateEventEnableMask Identifies which sensors within the composite state effector should have their operational state event message generation enabled after initialization <ul style="list-style-type: none"> [7] – 1b = enable event message generator for state sensor at offset 7 0b = disable event message generator for state sensor at offset 7 [6] – 1b = enable event message generator for state sensor at offset 6 0b = disable event message generator for state sensor at offset 6 ... [2] – 1b = enable event message generator for state sensor at offset 2 0b = disable event message generator for state sensor at offset 2 [1] – 1b = enable event message generator for state sensor at offset 1 0b = disable event message generator for state sensor at offset 1 [0] – 1b = enable event message generator for state sensor at offset 0 0b = disable event message generator for state sensor at offset 0
uint8	stateValue0 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.
uint8	stateValue1 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.
uint8	stateValue2 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.
	...
uint8	stateValue6 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.
uint8	stateValue7 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.

2903 **28.15 Effector Auxiliary Names PDR**

2904 The Effector Auxiliary Names PDR may be used to provide optional information that names an effector.
 2905 This record may be used for a single effector or multiple effectors if the effector is a composite state
 2906 effector.

2907 The nameLanguageTag field can be used to identify the language (such as French, Italian, or English)
 2908 that is associated with the particular effector name. Table 92 describes the format of this PDR.

2909 **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:	

28.16 OEM Effector Semantic PDR

The OEM Effector Semantic PDR is used to provide information about an OEM effector semantic used with one or more PLDM effectors that are represented in the PDRs. The information includes an ID for the vendor and a vendor-defined ID number for identifying the effector semantic. The PDR also allows one or more descriptive name strings to be provided for the vendor-defined effector semantic. The name strings may be provided in different character sets and languages.

The OEMEffectorSemanticHandle value in the PDR is used by other PDRs, such as the PLDM State Effector PDR, to point to the particular PLDM OEM Effector Semantic PDR within the PDR Repository. OEMStateSetIDHandle values are thus required to be unique for each different PLDM OEM State Set PDR within a given PDR Repository.

The OEMSemanticID field enables the vendor that defined the semantic to assign an ID value to its semantic. The OEMSemanticID field is thus defined relative to the given vendor ID.

The OEM Effector Semantic PDR also contains a PLDMTerminusHandle value. The PLDMTerminusHandle is used to provide a record of the terminus from which the PDR was imported. It is expected that most vendors will define their OEMSemanticID values in a global manner in which the ID has the same meaning regardless of the PLDMTerminusHandle value.

Table 93 describes the format of this PDR.

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]

28.17 Entity Association PDR

The Entity Association PDR is used to form associations between entities, such as physical and logical entities. See clause 10 for more information. Table 94 describes the format of this PDR.

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]

2932 **28.18 Entity Auxiliary Names PDR**

2933 The Entity Auxiliary Names PDR may be used to provide optional information that names a particular
 2934 instance of an entity. The PDR can also be used to name a particular range of instances of an entity,
 2935 provided that the instances share the same containerID.

2936 The nameLanguageTag field can be used to identify the language (such as French, Italian, or English)
 2937 that is associated with the particular entity name. Table 95 describes the format of this PDR.

2938 **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]

28.19 OEM EntityID PDR

The OEM EntityID PDR can be used to provide a vendor-specific EntityID definition when no PLDM predefined EntityID corresponds to the type of entity that the vendor wants to represent.

When the entityType value is in the OEM range of values, the EntityID portion of the entityType field is OEM-defined. The EntityID value is then used as an OEMEntityIDHandle to locate the corresponding OEM EntityID PDR.

OEM Entity Type PDRs need to be able to be exported by a terminus, such as a terminus on a hot-plug card. The numbers in a given vendor's Device PDRs must be picked a priori by the vendor. Thus, duplications may exist among the OEM EntityID values that different vendors choose. The Discovery Agent function is responsible for adjusting the OEM Entity Type values to resolve any conflicts that may occur when it integrates PDRs into the Primary PDR Repository. Users of OEM EntityID values must be aware that these values may differ between different PDR Repositories. That is, an OEM EntityID for "widget" from vendor "ABC" will not always have the same Entity ID value across PDRs.

To facilitate the identification of particular OEM EntityIDs from a given vendor, each PDR includes a vendor-specific ID value that does not get altered by the Discovery Agent function. When used in conjunction with the vendor's ID, this provides a value that can always be used to identify the particular vendor-defined EntityID definition.

Table 96 describes the format of this PDR.

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]

28.20 Interrupt Association PDR

The Interrupt Association PDR is used to form associations between interrupt source entities and interrupt target entities. See 11.10 for more information. Table 97 describes the format of this PDR.

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]

28.21 Event Log PDR

The Event Log PDR is used to describe characteristics of the PLDM Event Log (if implemented). The specification defines the existence of only a single, central PLDM Event Log function. Therefore, only one occurrence of a PLDM Event Log PDR shall exist in a Primary PDR Repository.

Table 98 describes the format of this PDR.

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>

2968 28.22 FRU Record Set PDR

2969 The FRU Record Set PDR is used to describe characteristics of the PLDM FRU Record Set Data defined
 2970 in [DSP0257](#). The information can be used to locate a Terminus that holds FRU Record Set Data in order
 2971 to access that data using the commands specified in [DSP0257](#). The PDR also identifies the particular
 2972 Entity that is associated with the FRU information.

2973 Table 99 describes the format of this PDR.

2974

2975

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.

2976

28.23 OEM Device PDR

2977 The OEM Device PDR can be used to provide OEM (vendor-specific) information. The OEM-specific data
 2978 portion in an OEM Device PDR is limited to a maximum size of 64 KB. Higher-level system specifications
 2979 may place additional limits on the size and number of OEM Device PDRs that may be supported in a
 2980 given PLDM subsystem implementation. An OEM Device PDR must have at least one byte of
 2981 VendorSpecificData.

2982 This type of PDR shall be copied by the Discovery Agent into the Primary PDR Repository dependent on
 2983 the setting of the copyPDR field. The PDR may also be preconfigured into the Primary PDR Repository.
 2984 That is, this PDR is not restricted to being only used or migrated from repositories that are separate from
 2985 the Primary PDR Repository.

2986 The OEM PDR is a slightly smaller version of the OEM Device PDR that can be used in situations where
 2987 it is not necessary or desired to associate the PDR to a particular terminus or have the information copied
 2988 from a Device PDR Repository into the Primary PDR Repository.

2989 Table 100 describes the format of this PDR.

28.23.1 Copy Behavior

2991 If the copyPDR parameter is set to copyToPrimaryRepository, the Discovery Agent shall overwrite any
 2992 pre-existing PDRs for the terminus that have the same vendorIANA and VendorHandle values.

28.23.2 Removal Behavior

2994 The OEM Device PDR is allowed to be removed from the Primary PDR Repository if the Discovery Agent
 2995 detects that the terminus that is associated with the PDR has been removed or is no longer available.

2996

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]

2997 **28.24 OEM PDR**

2998 The OEM PDR can be used to provide OEM (vendor-specific) information. The OEM-specific data portion
 2999 in an OEM PDR is limited to a maximum size of 64 KB. Higher-level system specifications may place
 3000 additional limits on the size and number of OEM PDRs that may be supported in a given PLDM
 3001 subsystem implementation. An OEM PDR must have at least one byte of vendorSpecificData. The OEM
 3002 Device PDR is an extended version of the OEM PDR that is used when it is necessary to associate the
 3003 PDR to a particular terminus or to have the information copied from a Device PDR Repository into the
 3004 Primary PDR Repository.

3005 Table 101 describes the format of this PDR.

3006 **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]

3007 28.25 Compact Numeric Sensor PDR

3008 The Compact Numeric Sensor PDR is designed for Management Controller (MC) monitoring of a
 3009 sophisticated PLDM terminus (device) where data conversion is not required. This sensor always reports
 3010 normalized integer values. Temperature and counting sensors are examples of sensor types that may be
 3011 defined by this PDR sensor type. Any mapping to an external management protocol is defined outside of
 3012 this specification.

3013 The commands, which specify a “raw value” such as SetSensorThresholds, GetSensorThresholds and
 3014 GetSensorReading, shall use the sensor’s (integer) value.

3015 This sensor is for simple numeric sensor reporting. For complex designs, the standard Numeric Sensor
 3016 PDR is retained and supported.

3017 **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	sensorNameStringByteLength 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.
enum8	baseUnit The base unit of the reading returned by this sensor. See 27.4 for more information. value: { see Table 74 }
sint8	unitModifier A power-of-10 multiplier for the baseUnit. See 27.4 for more information.
enum8	occurrenceRate 0 : No Occurrence Rate 1 : Per Microsecond 2 : Per Millisecond 3 : Per Second 4 : Per Minute 5 : Per Hour 6 : Per Day
bitfield8	rangeFieldSupport 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). [6:7] – reserved [5] – 1b = fatalLow field supported [4] – 1b = fatalHigh field supported [3] – 1b = criticalLow field supported [2] – 1b = criticalHigh field supported [1] – 1b = warningLow field supported [0] – 1b = warningHigh field supported
sint32	warningHigh 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.
sint32	warningLow 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.

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.

3018

3019 **28.26 Redfish Resource PDR**

3020 The Redfish Resource PDR provides the Redfish Schema information for every Redfish resource
3021 managed by a data provider. The usage of this PDR is defined in [DSP0218](#), *Platform Level Data Model*
3022 *for Redfish Device Enablement*.

3023 **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	ResourceFlags Flags associated with this Resource: [7:3] - reserved for future use [2] - is_collection; if 1b, this resource is a Redfish collection that contains zero or more resources sharing a common schema [1] - is_contained_in_collection; if 1b, the resource in which this resource is contained is a collection [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
uint32	ContainingResourceID 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. special value: 0x0000 0000 = "EXTERNAL". This value is used to identify the logical root of a device component's management topology. 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
strUTF-8	ProposedContainingResourceName 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> . 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.
uint16	SubURLLengthBytes Length in bytes of the SubURI path fragment (including the null terminator) for the primary resource

Type	Description
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, except as defined below for settings resources. Shall be a null byte if ContainingResourceID is EXTERNAL, except as defined below for settings resources.</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>Settings resources, may be expressed parallel to the resources to which they correspond by defining the SubURI appropriately. The SubURI for a settings object shall be formed by appending "/settings" to the SubURI for its corresponding resource. For a resource where the ContainingResourceID is EXTERNAL, the SubURI for the settings resource shall be expressed as "/settings"; this is the only case in which a leading slash ("/") is allowed for a SubURI.</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>
uint16	<p>AdditionalResourceIDCount</p> <p>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</p>
uint32	<p>AdditionalResourceID [0]</p> <p>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.</p>
uint16	<p>AdditionalResourceSubURLengthBytes [0]</p> <p>Length in bytes of the SubURI path fragment (including the null terminator) for this additional resource</p>
strUTF-8	<p>AdditionalResourceSubURI [0]</p> <p>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.</p>
...	...
uint32	<p>AdditionalResourceID [$N_A - 1$]</p> <p>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.</p>
uint16	<p>AdditionalResourceSubURLengthBytes [$N_A - 1$]</p> <p>Length in bytes of the SubURI path fragment (including the null terminator) for this additional resource</p>

Type	Description
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
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

3024 28.27 Redfish Entity Association PDR

3025 The Redfish Entity Association PDR provides the topology (or hierarchy) of Redfish (data) resources. The
 3026 usage of this PDR is defined in [DSP0218](#), *Platform Level Data Model for Redfish Device Enablement*.

3027 **Table 104 – Redfish Entity Association PDR format**

Type	Description
–	CommonHeader See 28.1.
<i>Container Entity Identification Information</i>	

Type	Description
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]
	...
uint32	ContainedEntityResourceID [$N_C - 1$]

3028 28.28 Redfish Action PDR

3029 The Redfish Action PDR provides the details of the "Actions" a resource can execute. The "Actions" are
 3030 described in standard Redfish resource schema definition. The usage of this PDR is defined in DSP0218
 3031 Platform Level Data Model for Redfish Device Enablement.

3032 **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.

Type	Description
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 the action excluding the ResourceType (as defined by DSP0266), null-terminated. For example, ActionName for the "Reset" action of a ComputerSystem using Redfish is "Reset".
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 Action within the resource's major dictionary. For a non-OEM specific action, this will usually be in the form "Actions/<QualifiedActionName>" (see DSP0266). For example, ActionPath for the "Reset" action of a ComputerSystem using Redfish is "Actions/ComputerSystem.Reset".
	...
uint8	ActionNameLengthBytes [N_A - 1] Including null terminator
utf8string	ActionName [N_A - 1] The name of the action excluding the ResourceType (as defined by DSP0266), 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 Action within the resource's major dictionary. For a non-OEM specific action this will usually be in the form "Actions/<QualifiedActionName>" (see DSP0266)

29 Timing

Table 106 defines timing values that are specific to this document.

Table 106 – Monitoring and control timing specifications

Timing specification	Symbol	Min	Max	Description
PDR record handle retention	MC1	30 sec	–	See 26.2.8.

30 PLDM Command numbers

Table 107 defines the PLDM command numbers used in the requests and responses for the PLDM monitoring and control commands defined in this specification.

Table 107 – Command numbers

#	Command	Reference
Terminus commands		
0x01	SetTID (PLDM type 0; see DSP0240)	See 16.1.
0x02	GetTID (PLDM type 0; 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
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.

#	Command	Reference
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

3040

ANNEX A (informative)

Change log

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
1.2.1	2021-MM-DD	Added clarification for behavior when attempting to set a threshold for a numeric sensor that is not settable; removed claim that it is possible to tell whether a numeric sensor threshold is settable. Added INVALID_DATA_TRANSFER_HANDLE completion Code in PollForPlatformEventMessage for when transfer handle is bad. Clarified scope of eventDataLength field in Redfish Message Event. Clarified that GetTID and SetTID are not PLDM Type 2 commands.

Version	Date	Description
		Added documentation for behavior in synchronyConfiguration setting when eventMessageGlobalEnable is disabled in the EventMessageSupported command. Documented support for Redfish Settings objects in Redfish Resource PDR SubURIs.
1.2.2	2022-10-26	Released as DMTF Standard

3046

3047

Bibliography

3048

3049 DMTF DSP4004, *DMTF Release Process 2.4*,
3050 https://dmf.org/sites/default/files/standards/documents/DSP4004_2.4.pdf

3051