

1 **EPCglobal** 

2 **Reader Management 1.0.1**

3 May 31, 2007

## 4 Abstract

5 This document defines Version 1.0 of the wire protocol used by management software to  
6 monitor the operating status and health of EPCglobal compliant RFID Readers. This  
7 document complements the EPCglobal Reader Protocol Version 1.1 specification [RP1].  
8 In addition, this document defines Version 1.0 of the EPCglobal SNMP RFID MIB.

## 9 Audience for this document

10 The target audience for this specification includes:

- 11 • EPC Middleware vendors
- 12 • Reader vendors
- 13 • Application developers
- 14 • Management Software developers
- 15 • System integrators

## 16 Status of this document

17 This section describes the status of this document at the time of its publication. Other  
18 documents may supersede this document. The latest status of this document series is  
19 maintained by EPCglobal. This version of the document provides minor updates to the  
20 SNMP since the Standard was first ratified. The table below summarizes changes to the  
21 document since the ratification of Reader Management 1.0.

Date and Version Number	Section(s)	Changes	Approved By
May 31, 2007 1.0.1	10.4.2	- Removed some inconsistencies between Abstract Model and MIB related to Alarm Controls (removed antenna-readpoint level suppression counters and controls and put them at the per-operation level where the abstract model calls for them  - Added EpcgAdministrativeStatus type  - Added ioValue enumeration to	

EpcgTriggerType		
December 5, 2006	All	Initial revision
1.0		

22

23 Comments on this document should be sent to the EPCglobal Software Action Group

24 Reader Management Working Group mailing list

25 [sag\\_rm\\_wg@lists.ecpcglobalinc.org](mailto:sag_rm_wg@lists.ecpcglobalinc.org).

## 26 **Table of Contents**

27	Abstract.....	2
28	Abstract.....	2
29	Audience for this document.....	2
30	Audience for this document.....	2
31	Status of this document.....	2
32	Status of this document.....	2
33	1 Introduction.....	11
34	2 Terminology.....	11
35	3 Protocol Layers.....	11
36	4 Object Model.....	14
37	4.1 Monitored Objects.....	14
38	4.1.1 ReaderDevice Object.....	15
39	4.1.2 CommandChannel.....	17
40	4.1.3 Source Object.....	17
41	4.1.4 ReadPoint Object.....	19
42	4.1.5 AntennaReadPoint Object.....	19
43	4.1.6 Trigger Object.....	20
44	4.1.7 IOPort Object.....	21
45	4.1.8 AlarmChannel Object.....	21
46	NotificationChannel Object.....	22
47	4.2 AlarmControl Objects.....	23
48	4.3 Alarm Objects.....	24
49	4.4 Enumerations.....	26
50	5 Reader Layer – Commands.....	26
51	5.1 ReaderDevice.....	28
52	5.1.1 ReaderDevice.getDescription.....	28
53	5.1.2 ReaderDevice.setDescription.....	29
54	5.1.3 ReaderDevice.getLocationDescription.....	29
55	5.1.4 ReaderDevice.setLocationDescription.....	30
56	5.1.5 ReaderDevice.getContact.....	31
57	5.1.5.1 ReaderDevice.setContact.....	31
58	5.1.6 ReaderDevice.getSerialNumber.....	32

59	5.1.7 ReaderDevice.getOperStatus.....	33
60	5.1.8 ReaderDevice.getOperStatusAlarmControl .....	33
61	5.1.9 ReaderDevice.getFreeMemory .....	34
62	5.1.10 ReaderDevice.getFreeMemoryAlarmControl .....	35
63	5.1.11 ReaderDevice.getNTPServers.....	36
64	5.1.12 ReaderDevice.getDHCPServer .....	36
65	5.1.13 ReaderDevice.getIOPort .....	37
66	5.1.14 ReaderDevice.getAllIOPorts.....	37
67	5.1.15 ReaderDevice.resetStatistics .....	38
68	5.1.16 ReaderDevice.removeAlarmChannels .....	39
69	5.1.17 ReaderDevice.removeAllAlarmChannels .....	40
70	5.1.18 ReaderDevice.getAlarmChannel.....	40
71	5.1.19 ReaderDevice.getAllAlarmChannels .....	41
72	5.2 NotificationChannel.....	42
73	5.2.1 NotificationChannel.getLastNotificationAttempt .....	42
74	5.2.2 NotificationChannel.getLastSuccessfulNotification .....	42
75	5.2.3 NotificationChannel.getOperStatus.....	43
76	5.2.4 NotificationChannel.setAdminStatus.....	44
77	5.2.5 NotificationChannel.getAdminStatus .....	44
78	5.2.6 NotificationChannel.getOperStatusAlarmControl .....	45
79	5.3 AlarmChannel.....	46
80	5.3.1 AlarmChannel.create.....	46
81	5.3.2 AlarmChannel.getName .....	47
82	5.3.3 AlarmChannel.getAddress .....	47
83	5.3.4 AlarmChannel.setAddress.....	48
84	5.4 ReadPoint .....	49
85	5.4.1 ReadPoint.getClassName .....	49
86	5.4.2 ReadPoint.getDescription.....	50
87	5.4.3 ReadPoint.setDescription .....	50
88	5.4.4 ReadPoint.getAdminStatus .....	51
89	5.4.5 ReadPoint.setAdminStatus.....	52
90	5.4.6 ReadPoint.getOperStatus.....	52
91	5.4.7 ReadPoint.getOperStatusAlarmControl .....	53

92	5.5 AntennaReadPoint .....	54
93	5.5.1 AntennaReadPoint.getIdentificationCount .....	54
94	5.5.2 AntennaReadPoint.getFailedIdentificationCount .....	55
95	5.5.3 AntennaReadPoint.getMemReadCount .....	55
96	5.5.4 AntennaReadPoint.getFailedMemReadCount .....	56
97	5.5.5 AntennaReadPoint.getFailedMemReadAlarmControl.....	57
98	5.5.6 AntennaReadPoint.getWriteCount.....	57
99	5.5.7 AntennaReadPoint.getFailedWriteCount.....	58
100	5.5.8 AntennaReadPoint.getFailedWriteAlarmControl .....	59
101	5.5.9 AntennaReadPoint.getKillCount.....	59
102	5.5.10 AntennaReadPoint.getFailedKillCount.....	60
103	5.5.11 AntennaReadPoint.getFailedKillAlarmControl .....	61
104	5.5.12 AntennaReadPoint.getEraseCount .....	61
105	5.5.13 AntennaReadPoint.getFailedEraseCount .....	62
106	5.5.14 AntennaReadPoint.getFailedEraseAlarmControl .....	62
107	5.5.15 AntennaReadPoint.getLockCount.....	63
108	5.5.16 AntennaReadPoint.getFailedLockCount.....	64
109	5.5.17 AntennaReadPoint.getTimeEnergized .....	66
110	5.5.18 AntennaReadPoint.getNoiseLevel .....	67
111	5.6 Source Object .....	68
112	5.6.1 Source.getUnknownToGlimpsedCount .....	68
113	5.6.2 Source.getGlimpsedToUnknownCount .....	69
114	5.6.3 Source.getGlimpsedToObservedCount.....	69
115	5.6.4 Source.getObservedToLostCount .....	70
116	5.6.5 Source.getLostToGlimpsedCount .....	70
117	5.6.6 Source.getLostToUnknownCount.....	71
118	5.6.7 Source.getOperStatus .....	72
119	5.6.8 Source.getAdminStatus .....	72
120	5.6.9 Source.setAdminStatus .....	73
121	5.6.10 Source.getOperStatusAlarmControl.....	74
122	5.7 Trigger Object.....	74
123	5.7.1 Trigger.getFireCount.....	74
124	5.8 IOPort Object.....	75

125	5.8.1 IOPort.getName.....	75
126	5.8.2 IOPort.getDescription .....	76
127	5.8.3 IOPort.setDescription .....	76
128	5.8.4 IOPort.getOperStatus .....	77
129	5.8.5 IOPort.getAdminStatus .....	78
130	5.8.6 IOPort.setAdminStatus.....	78
131	5.8.7 IOPort.getOperStatusAlarmControl.....	79
132	5.8.8 AlarmControl.getName .....	81
133	5.8.9 AlarmControl.setEnabled .....	82
134	5.8.10 AlarmControl.setEnabled.....	82
135	5.8.11 AlarmControl.getLevel .....	83
136	5.8.12 AlarmControl.setLevel.....	84
137	5.8.13 AlarmControl.getSuppressInterval.....	84
138	5.8.14 AlarmControl.setSuppressInterval .....	85
139	5.9 EdgeTriggeredAlarmControl.....	85
140	5.9.1 EdgeTriggeredAlarmControl.getAlarmThreshold .....	87
141	5.9.2 EdgeTriggeredAlarmControl.setAlarmThreshold.....	87
142	5.9.3 EdgeTriggeredAlarmControl.getRearmThreshold.....	89
143	5.9.4 EdgeTriggeredAlarmControl.setRearmThreshold .....	89
144	5.9.5 EdgeTriggeredAlarmControl.getDirection .....	90
145	5.9.6 EdgeTriggeredAlarmControl.setDirection .....	90
146	5.9.7 EdgeTriggeredAlarmControl.getStatus.....	91
147	5.10 TTOperationalStatusAlarmControl .....	92
148	5.10.1 TTOperationalStatusAlarmControl.getTriggerFromState .....	93
149	5.10.2 TTOperationalStatusAlarmControl.setTriggerFromState.....	94
150	5.10.3 TTOperationalStatusAlarmControl.getTriggerToState.....	94
151	5.10.4 TTOperationalStatusAlarmControl.setTriggerToState .....	95
152	6 Reader Layer – Alarm Notifications.....	96
153	6.1 Alarm Objects.....	96
154	6.2 Alarm.....	97
155	6.2.1 Alarm.getReaderDeviceEPC.....	99
156	6.2.2 Alarm.getReaderDeviceName.....	99
157	6.2.3 Alarm.getReaderDeviceHandle.....	100

158	6.2.4 Alarm.getReaderDeviceRole.....	100
159	6.2.5 Alarm.getTimeTicks .....	101
160	6.2.6 Alarm.getTimeUTC .....	101
161	6.2.7 Alarm.getName .....	102
162	6.2.8 Alarm.getAlarmLevel .....	102
163	6.2.9 Alarm.getSuppressCount.....	103
164	6.3 FreeMemoryAlarm .....	103
165	6.3.1 FreeMemoryAlarm.getFreeMemory .....	104
166	6.4 FailedWriteAlarm.....	104
167	6.4.1 FailedWriteAlarm.getReadPointName .....	105
168	6.4.2 FailedWriteAlarm.getFailedWriteCount.....	106
169	6.4.3 FailedWriteAlarm.getNoiseLevel .....	106
170	6.5 FailedEraseAlarm .....	107
171	6.5.1 FailedEraseAlarm.getReadPointName.....	107
172	6.5.2 FailedEraseAlarm.getFailedEraseCount .....	108
173	6.5.3 FailedEraseAlarm.getNoiseLevel .....	109
174	6.6 FailedKillAlarm.....	109
175	6.6.1 FailedKillAlarm.getReadPointName .....	110
176	6.6.2 FailedKillAlarm.getFailedKillCount.....	110
177	6.6.3 FailedKillAlarm.getNoiseLevel .....	111
178	6.7 FailedLockAlarm.....	111
179	6.7.1 FailedLockAlarm.getReadPointName .....	112
180	6.7.2 FailedLockAlarm.getFailedLockCount.....	113
181	6.7.3 FailedLockAlarm.getNoiseLevel .....	113
182	6.8 FailedMemReadAlarm .....	114
183	6.8.1 FailedMemReadAlarm.getReadPointName.....	115
184	6.8.2 FailedMemReadAlarm.getFailedMemReadCount.....	115
185	6.8.3 FailedMemReadAlarm.getNoiseLevel.....	116
186	6.9 TTOperStatusAlarm .....	117
187	6.9.1 TTOperStatusAlarm.getFromState .....	117
188	6.9.2 TTOperStatusAlarm.getToState.....	118
189	6.10 ReaderDeviceOperStatusAlarm.....	118
190	6.11 IOPortOperStatusAlarm .....	119



191	6.11.1 IOPortOperStatusAlarm.getIOPortName.....	119
192	6.12 ReadPointOperStatusAlarm.....	120
193	6.12.1 ReadPointOperStatusAlarm.getReadPointName .....	120
194	6.13 SourceOperStatusAlarm .....	121
195	6.13.1 SourceOperStatusAlarm.getSourceName .....	121
196	6.14 NotificationChannelOperStatusAlarm.....	122
197	6.14.1 NotificationChannelOperStatusAlarm.getNotificationChannelName .....	122
198	7 Enumerated types.....	123
199	7.1.1 AdministrativeStatus .....	123
200	7.1.2 OperationalStatus .....	123
201	7.1.3 EdgeTriggeredAlarmDirection.....	124
202	7.1.4 EdgeTriggeredAlarmStatus.....	124
203	7.1.5 AlarmLevel.....	124
204	8 Error Handling .....	125
205	8.1 Error Conditions .....	125
206	8.2 Communication Errors .....	125
207	8.2.1 Communication Host-to-Reader .....	125
208	8.2.2 Communication Reader-to-Host .....	125
209	8.3 Command Errors.....	126
210	9 Vendor Extensions.....	127
211	10 Message/Transport Bindings (MTBs).....	130
212	10.1 Address Notation .....	130
213	10.2 Vendor Extension Details.....	130
214	10.3 XML Message Format.....	131
215	10.3.1 Command XML Message Encoding (Host-To-Reader) .....	131
216	10.3.2 Reply XML Message Encoding (Reader-To-Host) .....	151
217	10.3.3 Alarm Notification XML Message Encoding (Reader-To-Host) .....	180
218	10.3.4 Common Data Formats .....	187
219	10.3.5 EPCglobal Standard Header.....	188
220	10.4 SNMP MIB.....	189
221	10.4.1 Vendor Extension Details.....	191
222	10.4.2 EPCglobal RFID Reader Management MIB.....	193
223	10.4.2.1 EPCglobal SMI MIB.....	193

224	10.4.2.2 EPCglobal Reader MIB .....	193
225	11 Acknowledgements.....	240
226	12 References.....	245
227		
228		

## 229 **1 Introduction**

230 This document defines Version 1.0 of the wire protocol used by management software to  
231 monitor the operating status and health of EPCglobal compliant tag Readers. This  
232 document complements the EPCglobal Reader Protocol Version 1.1 specification [RP1].  
233 In addition, this document defines Version 1.0 of the EPCglobal SNMP RFID MIB, and  
234 specifies the set of SNMP MIBII groups [MIBII] required to comply with this EPCglobal  
235 Reader Management Specification over SNMP. The terms “tag Reader” and “Reader”  
236 include RFID tag Readers, supporting any combination of RF protocols, fixed and hand-  
237 held, etc. It also includes Readers of other kinds of tags, such as bar codes. Tag Readers,  
238 despite the name, may also have the ability to write data into tag memory.

239 The Reader Management Protocol specifies the interaction between a device capable of  
240 interfacing with tags, and management software. These two parties are herein referred to  
241 as the Reader and the Host. The Host may be a fully featured Management Console  
242 capable of processing SNMP messages, or a dedicated application capable of  
243 communicating with the Reader to interface with RFID tags and monitor its health.

244 The collection of tag data between the Reader and the Host is defined in the EPCglobal  
245 Reader Protocol Version 1.1 [RP1]. This document focuses on the communication  
246 protocol required to monitor the health of the Reader.

## 247 **2 Terminology**

248 Within this specification, the terms SHALL, SHALL NOT, SHOULD, SHOULD NOT,  
249 MAY, NEED NOT, CAN, and CANNOT are to be interpreted as specified in Annex G of  
250 the ISO/IEC Directives, Part 2, 2001, 4th edition [ISODir2]. When used in this way,  
251 these terms will always be shown in ALL CAPS; when these words appear in ordinary  
252 typeface they are intended to have their ordinary English meaning.

253 All sections of this document are normative, with the exception of where it is explicitly  
254 noted as non-normative.

255 The following typographical conventions are used throughout the document:

- 256 • ALL CAPS type is used for the special terms from [ISODir2] enumerated above.
- 257 • Monospace type is used to denote programming language, UML, XML identifiers  
258 and other coding constructs, as well as for the text of XML documents.

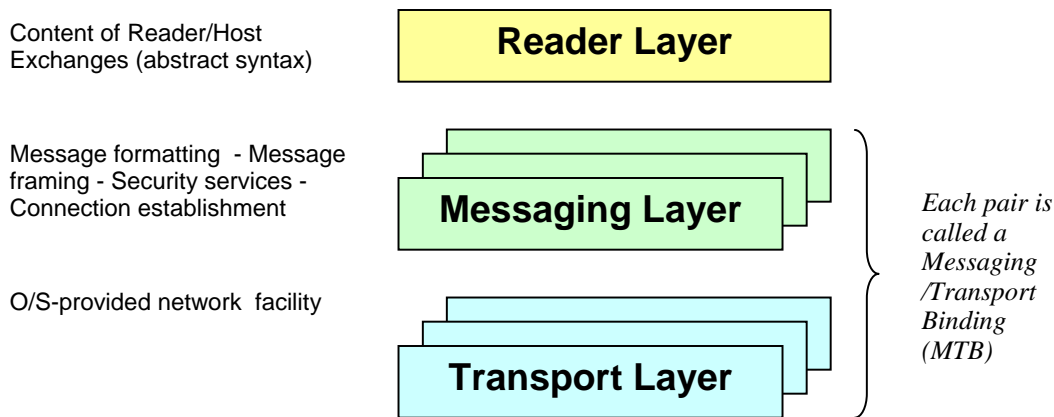
## 259 **3 Protocol Layers**

260 This document defines two separate but related management protocol specifications.

- 261 1. Specifies the EPCglobal SNMP MIB for monitoring the health of a Reader.
- 262 2. Specifies the EPCglobal Reader Management Protocol for monitoring the health  
263 of a Reader.

264 Number 2 follows the same layering structure defined by the EPCglobal Reader Protocol  
265 Version 1.1 [RP1]. The Reader Management Protocol is specified in three distinct layers,  
266 as illustrated below.

267



268

269 **Figure 1 Protocol Layers**

270

271 The layers are:

- 272 • *Reader Layer* This layer specifies the content and abstract syntax of messages  
273 exchanged between the Reader and Host. This layer is the heart of the Reader  
274 Protocol and the Reader Management Protocol, defining the operations that Readers  
275 expose to monitor their health.
- 276 • *Messaging layer* This layer specifies how messages defined in the Reader Layer are  
277 formatted, framed, transformed, and carried on a specific network transport. Any  
278 security services are supplied by this layer. (Examples of security services include  
279 authentication, authorization, message confidentiality, and message integrity.) The  
280 Messaging Layer specifies how an underlying network connection is established, any  
281 initialization messages required to establish synchronization or to initialize security  
282 services, and any processing such as encryption that is performed on each message.
- 283 • *Transport Layer* This layer corresponds to the networking facilities provided by the  
284 operating system or equivalent, and is specified elsewhere.

285 The Reader Management Protocol specification provides for multiple alternative  
286 implementations of the Messaging Layer. Each such implementation is called a  
287 Messaging/Transport Binding (MTB). Different MTBs provide for different kinds of  
288 transport, e.g., TCP/IP versus Bluetooth versus serial line. Different MTBs may also  
289 provide different kinds of security services, means for establishing connections (e.g.,  
290 whether the Reader contacts the Host or the Host contacts the Reader), and means for  
291 provisioning of configuration information.

292 This specification defines an XML MTB which complements that defined by the Reader  
293 Protocol. The Transport details can be found in the Reader Protocol Specification 1.1.  
294 The SNMP MIB components are also defined in detail in this specification.

295

296

297

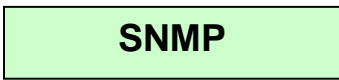
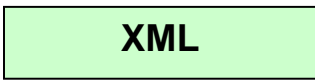
298

299

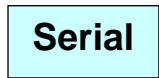
Content of Reader/Host Exchanges (abstract syntax)



Message formatting - Message framing - Security services - Connection establishment



O/S-provided network transport facility



300

301 **Figure 2 Protocol Layers Mapping**

302

303 The SNMP MIB is an example of another Message/Transport binding. The MIB is a  
 304 structuring and representation of Reader Object Model elements that conforms to the  
 305 SNMP specification “Structure of Management Information Version 2 (SMIv2)” [STD  
 306 58, RFC 2578]. The SNMP protocol has a well defined messaging protocol and transport  
 307 layer for getting and setting information, event notification, and security facilities. The  
 308 figure above depicts two MTB examples. On the left, the Reader Protocol Message  
 309 Format using XML as input and output, and TCP as the transport. On the right, SNMP  
 310 defining the Message formatting and UDP as the message transport. Note that the same  
 311 Reader Management Command Set applies to both MTBs.

312 The interface between the Reader Layer and the Messaging Layer is defined in terms of  
 313 message channels, each representing an independent communication between Reader and  
 314 Host. The Reader Management Protocol leverages the message channels defined in the  
 315 EPCglobal Reader Protocol Specification 1.1:

- 316 • *Command Channel* The command channel carries requests issued by the Host to the  
 317 Reader, and responses to these requests carried from the Reader to the Host. All  
 318 messages exchanged on the command channel follow this request/response pattern.  
 319 Most configuration interaction between Reader and Host takes place through the  
 320 command channel. The Reader MAY support one or more *Command Channels*.
- 321 • *Alarm Channel(s)* The alarm channel carries messages issued asynchronously by  
 322 the Reader to the Host. Messages on the alarm channel only flow to the Host. The  
 323 alarm channel is primarily used to support a mode of operation in which the Reader  
 324 asynchronously delivers health and status alerts to the Host, The Reader controls  
 325 when alarm channel messages are sent to the Host; the Host does not request alarm  
 326 channel messages. The Reader MAY support one or more Alarm Channels for the  
 327 delivery of Management Alarms.

328 • *Notification Channel(s)* In addition, the Reader MAY support one or more  
329 Notification Channels for delivery of tag data, as specified in the Reader Protocol  
330 Specification 1.1.

331 The Command channel provides the means to set and get the state, status, or value  
332 of attributes of a given object. Under certain conditions, the end-user may choose  
333 to disable the setter capabilities, therefore a Reader MAY provide the means to  
334 globally disable all *set* methods for all objects. If a set method is invoked after it  
335 has been disabled, the ERROR\_AUTHORIZATION SHALL be returned.

## 336 4 Object Model

337 The Reader Management Specification defines a set of conceptual objects and operations  
338 (a command set) which enables the Host to query the status of these objects in a standard  
339 way. This is described using an object model. The Reader Management specification  
340 shares a common object model with the Reader Protocol Specification 1.1; however some  
341 objects and attributes are not relevant to the Reader Management specification. Figure 3  
342 Reader Monitored Objects UML **Error! Reference source not found.**Figure 3 Reader  
343 Monitored Objects UML**Error! Reference source not found.**through

344 **Figure 14 Enumerations UML**

345

346 Figure 14 Enumerations UML present all of the objects in order to provide a complete  
347 model. Only the objects, attributes, and methods relevant to Reader Management are  
348 discussed in detail in this specification. Objects that are not relevant to this specification  
349 are shown in gray. Object attributes and methods from the Reader Protocol Specification  
350 1.1 and this Reader Management specification are shown in the detailed UML diagrams  
351 in Figure 4 ReaderDevice Object UML, Figure 5 Source Object UML, Figure 6  
352 ReadPoint Object UML, Figure 8 Trigger Object UML, and Figure 11  
353 NotificationChannel Object UML. The specific commands defined by this Reader  
354 Management specification are listed in detail in Section 5 Reader Layer – Commands,  
355 section 6 Reader Layer – Alarm Notifications, and section 7 Enumerated types of this  
356 document. Refer to the Reader Protocol Specification 1.1 for a description of the  
357 attributes and commands specific to that specification. A compliant Reader NEED NOT  
358 implement an object model internally, as long as the command messages can be correctly  
359 interpreted and executed, and alarms be correctly generated.

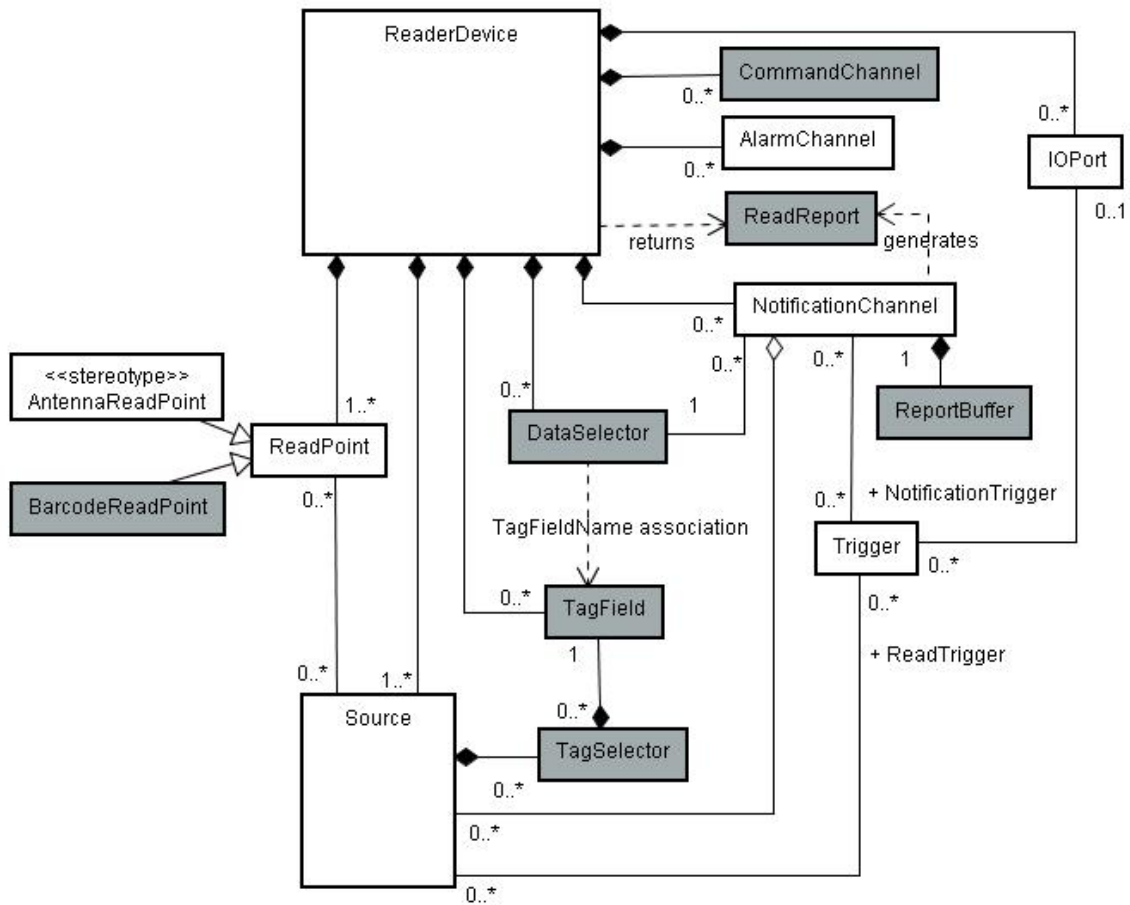
360 The health of a Reader can be monitored by actively querying Reader for the status of the  
361 various monitored objects, and/or by setting asynchronous notification alarms. Each  
362 object maintains a set of counters and status monitors that can be queried synchronously.  
363 Some objects also implement alarms that are configured to be triggered on a given set of  
364 conditions. The AlarmControl Objects define when alarms are generated. The Alarm  
365 Objects represent the actual content of the notification or alarm. AlarmControl Objects  
366 are defined in Section 4.2. Alarm Objects are defined in Section 4.3.

367 The following section provides a brief description of the major objects and their  
368 relationship to each other. For details on the objects and their operations, refer to Section  
369 5: Reader Layer – Commands

370  
371  
372  
373  
374  
375  
376  
377  
378

## 4.1 Monitored Objects

The following sections provide a brief description of the conceptual objects. Each object exposes a set of operations which are described in detail throughout this specification. Figure 3 Reader Monitored Objects UML illustrates the high level relationship between the components. The list of attributes and commands defined for each object are illustrated in each of the sections describing the objects. Objects with no visible attributes and/or commands, all in white, are illustrated in further detail in a subsequent UML diagram in the specification.



379  
380  
381

Figure 3 Reader Monitored Objects UML

### 4.1.1 ReaderDevice Object

The ReaderDevice object is the base container for most other objects in the Reader object model including at least one pre-configured CommandChannel object. The ReaderDevice object also contains several attributes that are used to manage the Reader device and, for network Readers, attributes used to manage the Reader's network interface.



388

389 **Figure 4 ReaderDevice Object UML**



390 **4.1.2 CommandChannel**

391 There are no explicit operations on this object. A compliant Reader MAY have one or  
392 more Command Channels with a default address where it is listening to incoming  
393 commands. If implemented, this default address SHALL be listed in the documentation  
394 of a compliant reader.

395 **4.1.3 Source Object**

396 Sources read tags and present their data to the Data Acquisition stage of the Read  
397 Subsystem in a Reader. Two examples of Sources are an RF antenna and a barcode  
398 scanner. Source objects are associated to the ReaderDevice object. One or more  
399 Sources MAY be associated with a NotificationChannel. In the Reader object  
400 model, a Source object can encapsulate one or more input sensors called  
401 ReadPoints. Refer to the Reader Protocol Specification 1.1 for further detail.

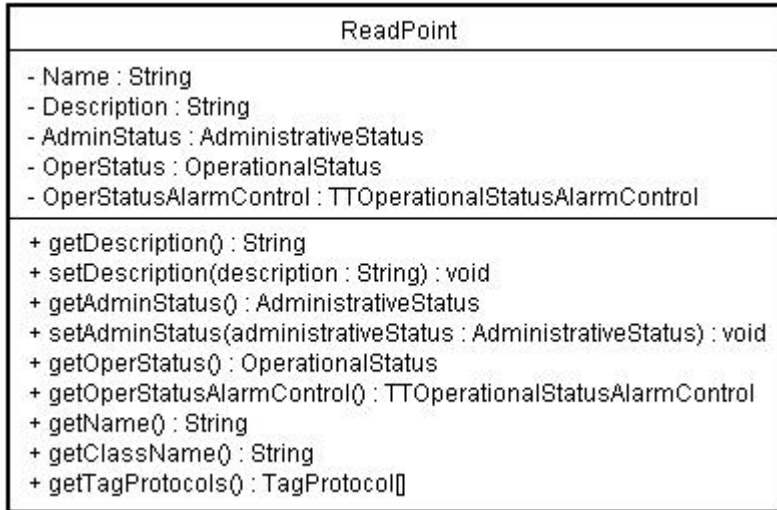
Source
<pre> - Name : String - ReadCyclesPerTrigger : int - ReadDutyCycle : int - ReadTimeout : int - GlimpsedTimeout : int - ObservedThreshold : int - ObservedTimeout : int - LostTimeout : int - UnknownToGlimpsedCount : int - GlimpsedToUnknownCount : int - GlimpsedToObservedCount : int - ObservedToLostCount : int - LostToGlimpsedCount : int - LostToUnknownCount : int - AdminStatus : AdministrativeStatus - OperStatus : OperationalStatus - OperStatusAlarmControl : TTOperationalStatusAlarmControl - Session : int  + getUnknownToGlimpsedCount() : int + getGlimpsedToUnknownCount() : int + getGlimpsedToObservedCount() : int + getObservedToLostCount() : int + getLostToObservedCount() : int + getLostToUnknownCount() : int + getOperStatus() : OperationalStatus + setAdminStatus(administrativeStatus : AdministrativeStatus) : void + getAdminStatus() : AdministrativeStatus + getOperStatusAlarmControl() : TTOperationalStatusAlarmControl + create(name : String) : Source + getName() : String + isReadOnly() : boolean + addReadPoints(readPoints : ReadPoint[]) : void + removeReadPoints(readPoints : ReadPoint[]) : void + removeAllReadPoints() : void + getReadPoint(name : String) : ReadPoint + getAllReadPoints() : ReadPoint[] + addReadTriggers(triggers : Trigger[]) : void + removeReadTriggers(triggers : Trigger[]) : void + removeAllReadTriggers() : void + getReadTrigger(name : String) : Trigger + getAllReadTriggers() : Trigger[] + addTagSelectors(selectors : TagSelector[]) : void + removeTagSelectors(selectors : TagSelector[]) : void + removeAllTagSelectors() : void + getTagSelector(name : String) : TagSelector + getAllTagSelectors() : TagSelector[] + getGlimpsedTimeout() : int + setGlimpsedTimeout(timeout : int) : void + getObservedThreshold() : int + setObservedThreshold(timeout : int) : void + getObservedTimeout() : int + setObservedTimeout(timeout : int) : void + getLostTimeout() : int + setLostTimeout(timeout : int) : void + rawReadIDs(dataSelector : DataSelector) : ReadReport + readIDs(dataSelector : DataSelector) : ReadReport + writeID(ID : binary, lockcode : binary) : void + kill(tagSelector : TagSelector[], data : TagFieldValue[]) : void + getReadCyclesPerTrigger() : int + setReadCyclesPerTrigger(cycles : int) : void + getReadDutyCycle() : int + setReadDutyCycle(dutyCycle : int) : void + getReadTimeout() : int + setReadTimeout(timeout : int) : void + write(tagSelector : TagSelector[], data : TagFieldValue[]) : void + read(tagSelector : TagSelector[], dataSelector : DataSelector) : ReadReport + getSession() : int + setSession(session : int) : void </pre>

402

403 **Figure 5 Source Object UML**

404 **4.1.4 ReadPoint Object**

405 A ReadPoint can be any physical entity that is capable of acquiring data. A single RF  
406 tag reader antenna is a simple example of a ReadPoint. In this version of the  
407 specification, the only supported ReadPoint is an AntennaReadPoint. Refer to  
408 the Reader Protocol Specification 1.1 for further detail.

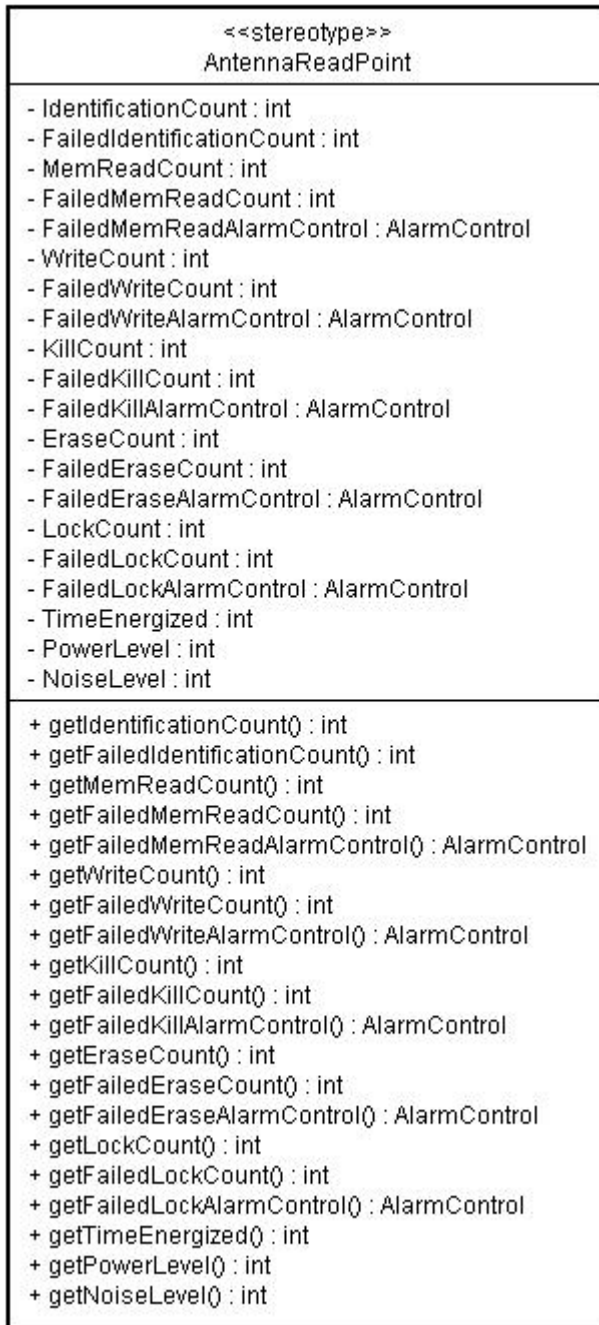


409

410 **Figure 6 ReadPoint Object UML**

411 **4.1.5 AntennaReadPoint Object**

412 AntennaReadPoints are ReadPoints which are physically implemented using  
413 antennas. Each AntennaReadPoint maintains a number of statistics to assist in  
414 monitoring the health of the reader. Refer to the Reader Protocol Specification 1.1 for  
415 further detail.

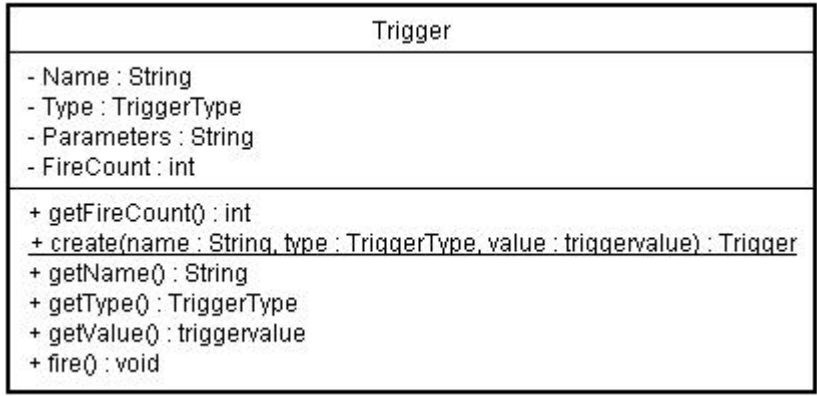


416

417 **Figure 7 AntennaReadPoint Object UML**

#### 418 **4.1.6 Trigger Object**

419 Trigger is a non-mutable object that can exist as a composite member of either a  
 420 Source or a NotificationChannel object, i.e., Trigger objects are added and  
 421 removed from Source and NotificationChannel objects. Refer to the Reader  
 422 Protocol Specification 1.1 for further detail.

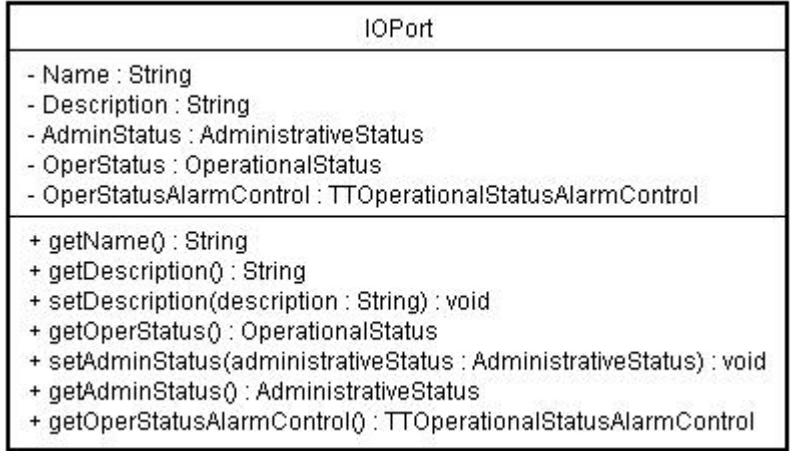


423

424 **Figure 8 Trigger Object UML**

425 **4.1.7 IOPort Object**

426 An IOPort provides the description of the hardware element that provides external  
 427 input and output lines to connect to other components outside the reader device. A  
 428 hardware vendor is free to define an IOPort as it best fits its needs. For example, one  
 429 vendor may choose to define a single IOPort for a 9-pin RS232 connector used to  
 430 interface with light switches or motion sensors. A second vendor may choose to assign a  
 431 unique IOPort to each of the pins, if it is capable of configuring and monitoring the  
 432 state of each pin independently. If a reader device exposes its IOPorts, it SHALL  
 433 implement the methods identified as such. Refer to the Reader Protocol Specification 1.1  
 434 for further detail.



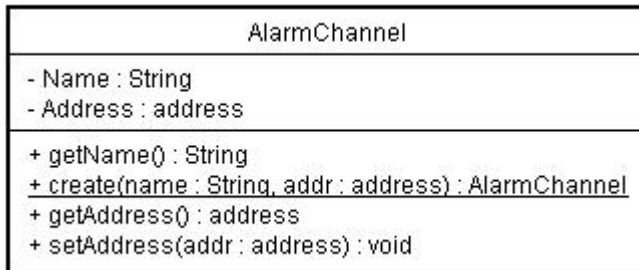
435

436 **Figure 9 IOPort Object UML**

437 **4.1.8 AlarmChannel Object**

438 The AlarmChannel carries messages issued asynchronously by the Reader to the Host.  
 439 Messages on an AlarmChannel only flow in this direction. AlarmChannels MAY  
 440 be implemented if a Reader is capable of notifying a host of changes in its health status.

441 It is strongly advised, but not required, to create/define AlarmChannels using the  
442 commands specified in this specification. A compliant system SHALL provide the  
443 means to create/ define AlarmChannels either using the commands specified in this  
444 specification, or through a Reader specific means, such as through the use of an out-of-  
445 band protocol, or via a configuration file.



446

447 **Figure 10 AlarmChannel Object UML**

### 448 **NotificationChannel Object**

449 A NotificationChannel object is responsible for communicating information to a  
450 host asynchronously. One or more Sources MAY be associated with a  
451 NotificationChannel. A DataSelector determines what data is reported to the host. Refer  
452 to the Reader Protocol Specification 1.1 for further detail.



453

454 **Figure 11 NotificationChannel Object UML**

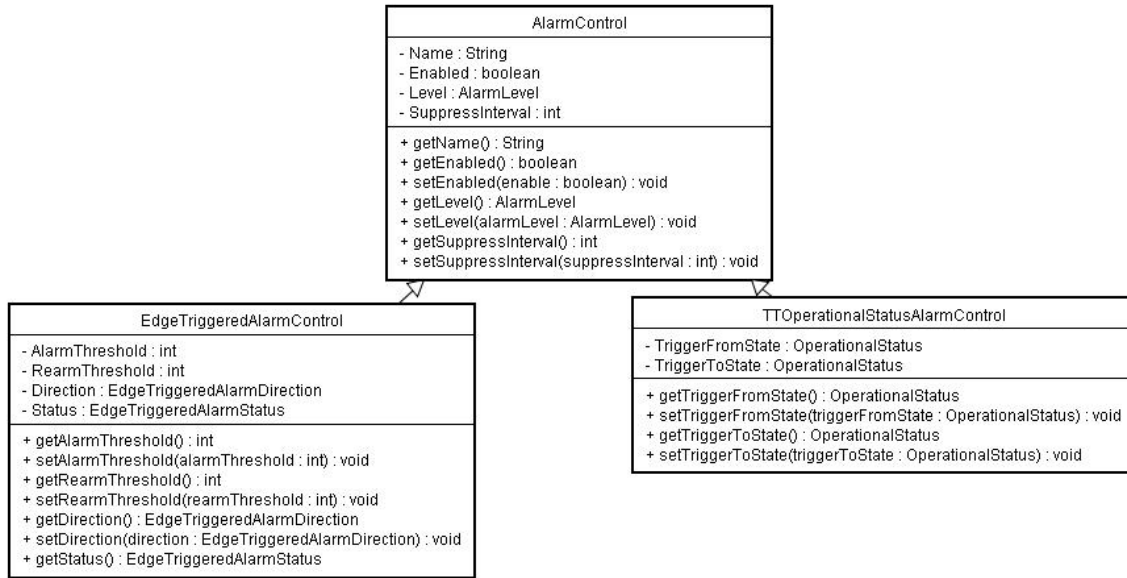
## 455 **4.2 AlarmControl Objects**

456 AlarmControl is the base class for all classes responsible for controlling the  
 457 generation of Alarms. AlarmControls SHALL be implemented if  
 458 AlarmChannels are implemented. In order to configure an Alarm on an object, the  
 459 monitoring host needs to obtain a reference to the AlarmControl and manipulate its  
 460 attributes. An Alarm object represents the actual notification sent to the monitoring  
 461 host. The UML for the AlarmControl objects is illustrated in Figure 12 Alarm  
 462 Control UML.

463 There are two types of AlarmControls defined by the subclasses:

- 464 • EdgeTriggeredAlarmControl
- 465 •
- 466 • TTOperationalStatusAlarmControl

467  
468



469  
470  
471

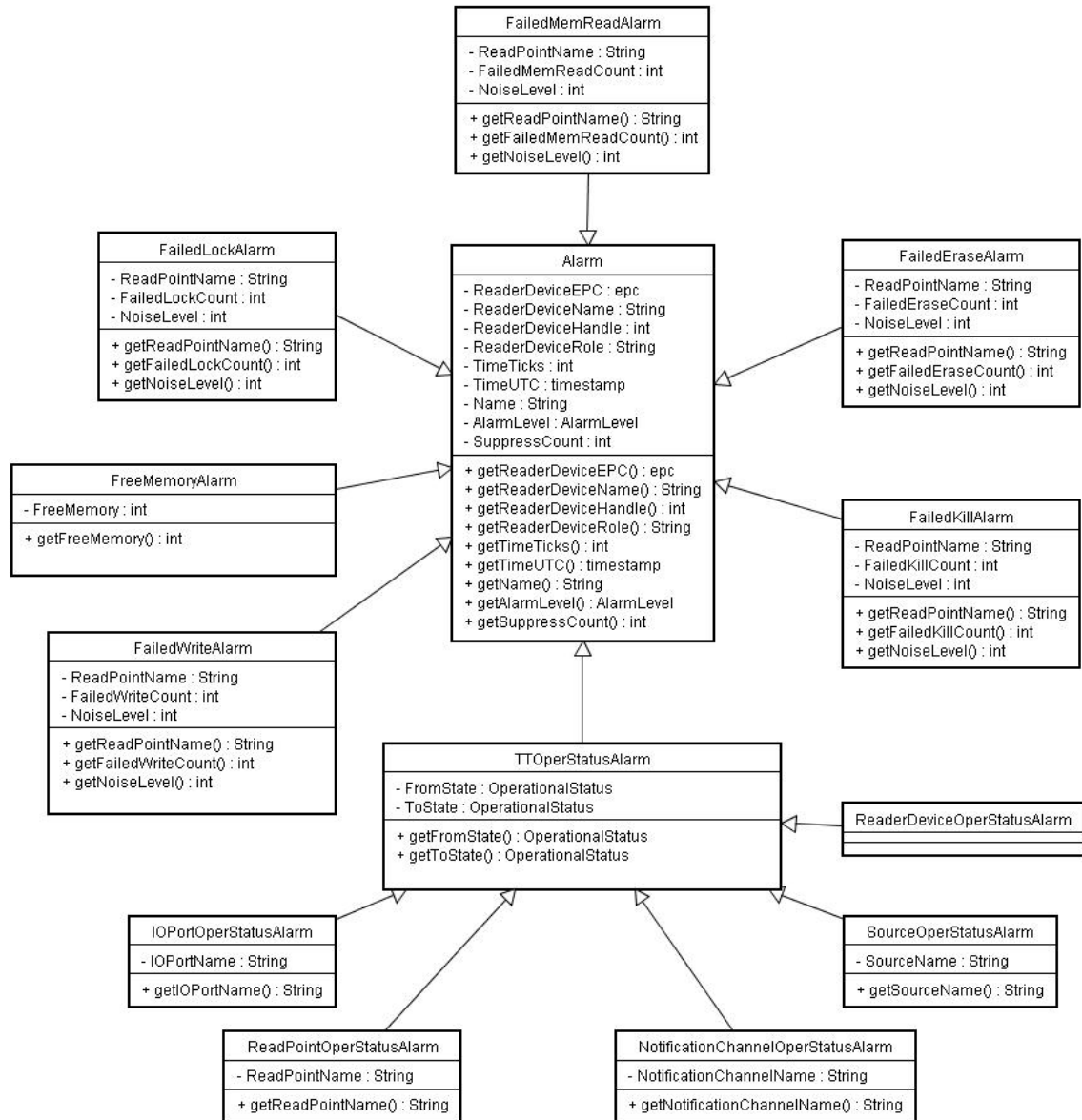
**Figure 12 Alarm Control UML**

### 4.3 Alarm Objects

473 The alarms themselves are messages that are sent to management systems and host  
474 applications. Alarm objects specify, in an abstract manner, the contents of these alarm  
475 messages. These Alarm objects do not specify persistent data objects maintained by the  
476 readers; rather they are instantiated in the course of a reader's generation or a  
477 management system's receipt and processing of alarms. Alarm objects SHALL be  
478 implemented if AlarmChannels are implemented. The UML for the Alarm objects is  
479 illustrated in Figure 13 Alarm UML.

480





481

482 **Figure 13 Alarm UML**

483

484

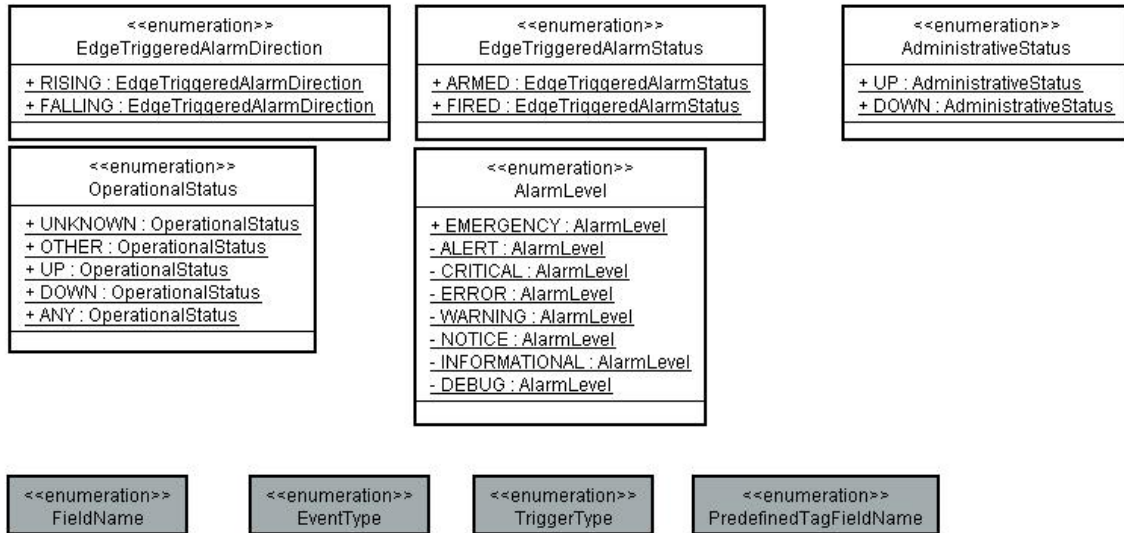
#### 485 **4.4 Enumerations**

486 The Enumerations contain sets of values used by other objects, such as the administrative  
 487 Status, Operational Status, Alarm levels, the status of EdgeTriggered Alarms, directions  
 488 of an alarm value, and the list of operators used to compare alarm levels. The UML for  
 489 the Enumerations is illustrated in

490 **Figure 14 Enumerations UML**

491

492 Figure 14 Enumerations UML.



493

494

495 Figure 14 Enumerations UML

496

497

498

## 499 5 Reader Layer – Commands

500 This Reader Management Specification defines a set of operations (a command set)  
501 which enables the Host to query the status of these objects in a standard way.

502 This section specifies messages exchanged at the Reader Layer in order to monitor the  
503 health of the Reader.

504 Messages on the command channel follow a request/response pattern, where the Host  
505 sends a request message, and the Reader responds later with a response message (be it a  
506 normal response or error response). Messages on the alarm channel are sent  
507 asynchronously from Reader to Host.

508 Reader Layer messages fall into the main functional groups defined by the objects in the  
509 previous section. The messages described in this specification are complementary to  
510 those defined in the Reader Management section of the Reader Protocol Version 1.1  
511 Specification.

512 Commands are described below in an abstract manner using a UML notation.  
513 This notation is identical to that used by the Reader Protocol Version 1.1.

514 Commands fall into three categories:

- 515 • DO-type commands that cause the Reader to take action.

- 516
- SET-type commands that cause the Reader to change internal state variables.
- 517
- GET-type commands that cause the Reader to return data to the Host on the
- 518
- command channel.

519

520 In short, ‘attributes’ or ‘properties’ (aka variables) with SET and/or GET  
521 operations, and ‘functions’ or ‘methods’ performing some action.

522 All commands are *atomic*, meaning that they SHALL be executed either  
523 completely or not at all. For example, when adding arrays of things (e.g.,  
524 `FieldNames` to a `DataSelector`), and one value is not supported or not  
525 known, then no values SHALL be added and error SHALL be raised.

526 The return values of the command indicate the return values if the command was  
527 successfully executed. If the command cannot be executed correctly, the return  
528 value SHALL be undefined and an error condition SHALL be raised. The error  
529 conditions are explained in Section 8 Error Handling.

530

531 **UML-like notation for an abstract command:**

```
532 [static] OBJECT.COMMAND ([  
533     PARAMETER: DataType,  
534     PARAMETER: DataType,  
535     ...]) : DataType
```

536

537 The `static` keyword indicates that a command doesn’t require an object  
538 instance in order to be executed.

539

540 **DataTypes for command parameters and command return values:**

- 541
- **address**  
542 See section 10.1.
- 543
- **epc**  
544 An EPC. The formatting depends on the MTB; it MAY be a string in urn-notation or  
545 also the raw binary value.
- 546
- 547
- 548
- **integer**  
549 All numeric values are 32-bit signed integers.
- 550
- **string**  
551 All strings SHALL be capable of being represented with UTF-8 encoding.
- 552
- **timestamp**

553 A timestamp (date and time) with millisecond accuracy. The exact format depends on  
554 the MTB in use.

555 • **type[]**

556 A list of the specified data type. The list is unordered unless explicitly stated  
557 otherwise.

558 • **void**

559 No data, indicating no parameters or return value (depending on where it's used).

560

561 In addition, many commands have object types as parameter or return value. The MTB  
562 defines how objects are encoded in the command and reply messages. The following  
563 section defines the Reader Management Command Set. A list of allowed errors is listed  
564 for each command. Section 8 Error Handling describes the Error conditions in detail.

565

## 566 **5.1 ReaderDevice**

### 567 **5.1.1 ReaderDevice.getDescription**

568 The Host queries the Reader for its user defined description. This is a text string that  
569 describes the general use of this particular reader.

570

571 **Compliance Requirement:** Compliant systems SHALL implement this command.

572

573 **Usage:**

574 `ReaderDevice.getDescription (void): string`

575

576 **Parameter(s):**

577 Data Type: `void`. This command takes no parameters.

578

579 **Return Value(s):**

580 Data Type: `string`. Reader will return its user defined description.

581

582 **Possible Error Conditions:**

583

584 `ERROR_UNKNOWN`

585

586

587 **5.1.2 ReaderDevice.setDescription**

588 The Host sets the Readers user-defined description. This is a text string that describes the  
589 general use of this particular reader.

590

591 **Compliance Requirement:** Compliant systems SHALL implement this command.

592

593 **Usage:**

594 `ReaderDevice.setDescription (description: string): void`

595

596 **Parameter(s):**

597 `description` – Data Type: `string`. A text string that describes the general use of  
598 this particular reader.

599

600 **Return Value(s):**

601 Data Type: `void`. This command will not return a value.

602

603 **Possible Error Conditions:**

604

605 `ERROR_PARAMETER_MISSING`

606 `ERROR_PARAMETER_INVALID_DATATYPE`

607 `ERROR_PARAMETER_LENGTH_EXCEEDED`

608 `ERROR_UNKNOWN`

609 `ERROR_AUTHORIZATION`

610

611

612 **5.1.3 ReaderDevice.getLocationDescription**

613 The Host queries the Reader for its user defined location description. This is a textual  
614 description of the location of the reader device.

615

616 **Compliance Requirement:** Compliant systems SHALL implement this command.

617

618 **Usage:**

619 `ReaderDevice.getLocationDescription (void): string`

620

621 **Parameter(s):**

622 Data Type: `void`. This command takes no parameters.

623

624 **Return Value(s):**

625 Data Type: `string`. Reader will return its user defined location description.

626

627 **Possible Error Conditions:**

628

629 `ERROR_UNKNOWN`

630

631

632 **5.1.4 ReaderDevice.setLocationDescription**

633 The Host sets the Readers user-defined location description. This is a textual description  
634 of the location of the reader device.

635

636 **Compliance Requirement:** Compliant systems SHALL implement this command.

637

638 **Usage:**

639 `ReaderDevice.setLocationDescription (locationDescription:  
640 string): void`

641

642 **Parameter(s):**

643 `locationDescription` – Data Type: `string`. A textual description of the  
644 location of the reader device.

645

646 **Return Value(s):**

647 Data Type: `void`. This command will not return a value.

648

649 **Possible Error Conditions:**

650

651 `ERROR_PARAMETER_MISSING`

652 `ERROR_PARAMETER_INVALID_DATATYPE`

653 ERROR\_PARAMETER\_LENGTH\_EXCEEDED

654 ERROR\_UNKNOWN

655 ERROR\_AUTHORIZATION

656

### 657 **5.1.5 ReaderDevice.getContact**

658 The Host queries the Reader for its user defined contact description. This information  
659 identifies the individual or organization responsible for administering the Reader.

660 **Compliance Requirement:** Compliant systems SHALL implement this command if, and  
661 only if, the system implements setContact.

662

#### 663 **Usage:**

664 `ReaderDevice.getContact (void): string`

665

#### 666 **Parameter(s):**

667 Data Type: `void`. This command takes no parameters.

668

#### 669 **Return Value(s):**

670 Data Type: `string`. Reader will return a contact information to the person responsible  
671 for the device or someone acting for him/her.

672

#### 673 **Possible Error Conditions:**

674 ERROR\_COMMAND\_NOT\_SUPPORTED

675 ERROR\_UNKNOWN

676 For a detailed error description see Reader Protocol chapter 8 Error Handling.

677

678

### 679 **5.1.5.1 ReaderDevice.setContact**

680 The Host sets the Readers user-defined contact description. This information identifies  
681 the individual or organization responsible for administering the Reader..

682

683

684 **Compliance Requirement:** Compliant systems SHALL implement this command.

685

686 **Usage:**

687 `ReaderDevice.setContact (contact: string): void`

688

689 **Parameter(s):**

690 `contact` – Data Type: `string`. A contact information to the person responsible for  
691 the device or someone acting for him/her.

692

693 **Return Value(s):**

694 Data Type: `void`. This command will not return a value.

695

696 **Possible Error Conditions:**

697 `ERROR_COMMAND_NOT_SUPPORTED`

698 `ERROR_PARAMETER_MISSING`

699 `ERROR_PARAMETER_INVALID_DATATYPE`

700 `ERROR_PARAMETER_LENGTH_EXCEEDED`

701 `ERROR_UNKNOWN`

702 `ERROR_AUTHORIZATION`

703

### 704 **5.1.6 ReaderDevice.getSerialNumber**

705 The Host queries the Reader for its serial number. This is the serial number of the reader  
706 device. Note that normally the serial number is also an integral part of the EPC. Since the  
707 serial number is set by the manufacturer, there is no `setSerialNumber()` command.

708

709 **Compliance Requirement:** Compliant systems SHALL implement this command.

710

711 **Usage:**

712 `ReaderDevice.getSerialNumber (void): string`

713

714 **Parameter(s):**

715 Data Type: `void`. This command takes no parameters.

716

717 **Return Value(s):**

718 Data Type: `string`. Reader will return the serial number of the reader device.



719

720 **Possible Error Conditions:**

721 ERROR\_UNKNOWN

722 For a detailed error description see Reader Protocol chapter 8 Error Handling.

723

724

### 725 **5.1.7 ReaderDevice.getOperStatus**

726 The Host queries the Reader for the ReaderDevice's OperStatus attribute, which  
727 is of type OperationalStatus.

728

729 **Compliance Requirement:** Compliant systems SHALL implement this command.

730

731 **Usage:**

732 ReaderDevice.getOperStatus (void): OperationalStatus

733

734 **Parameter(s):**

735 Data Type: void. This command takes no parameters.

736

737 **Return Value(s):**

738 Data Type: OperationalStatus. Reader will return the ReaderDevice object's  
739 OperStatus attribute.

740

741 **Possible Error Conditions:**

742

743 ERROR\_UNKNOWN

744

### 745 **5.1.8 ReaderDevice.getOperStatusAlarmControl**

746 The Host queries the ReaderDevice for its OperStatusAlarmControl attribute,  
747 of type TOperationalStatusAlarmControl. This attribute is the object that  
748 controls the conditions for generating alarms alerting a manager of changes in a  
749 ReaderDevice's operational status.

750

751 **Compliance Requirement:** Compliant systems SHALL implement this command if, and  
752 only if, AlarmChannels are implemented.

753

754 **Usage:**

755 `ReaderDevice.getOperStatusAlarmControl (void):`  
756 `TTOperationalStatusAlarmControl`

757

758 **Parameter(s):**

759 Data Type: `void`. This command takes no parameters.

760

761 **Return Value(s):**

762 Data Type: `TTOperationalStatusAlarmControl`. The `ReaderDevice` will  
763 return its `OperStatusAlarmControl` attribute.

764

765 **Possible Error Conditions:**

766 `ERROR_COMMAND_NOT_SUPPORTED`

767 `ERROR_UNKNOWN`

768

### 769 **5.1.9 ReaderDevice.getFreeMemory**

770 The Host queries the Reader for the amount of free memory available on the  
771 `ReaderDevice`. The `FreeMemory` attribute represents the number of kilobytes (KB)  
772 of Read/Write memory available to internal or external resources for buffering or other  
773 processing of information. For example, the `FreeMemory` attribute may include the  
774 number of kilobytes available for building a `ReportBuffer` (see RP1).

775

776 **Compliance Requirement:** Compliant systems MAY implement this command.

777

778 **Usage:**

779 `ReaderDevice.getFreeMemory (void): integer`

780

781 **Parameter(s):**

782 Data Type: `void`. This command takes no parameters.

783

784 **Return Value(s):**

785 Data Type: `integer`. Reader will return its available free memory specified in kilo  
786 bytes (KB).

787

788 **Possible Error Conditions:**

789 ERROR\_COMMAND\_NOT\_SUPPORTED

790 ERROR\_UNKNOWN

791 For a detailed error description see Reader Protocol chapter 8 Error Handling.

792

793

### 794 **5.1.10 ReaderDevice.getFreeMemoryAlarmControl**

795 The Host queries the Reader for the ReaderDevice object's  
796 FreeMemoryAlarmControl attribute, of type EdgeTriggeredAlarmControl.  
797 This object controls the generation of alarms alerting a managing system of a reduction of  
798 a reader's free memory resources below a specified threshold.

799

800 **Compliance Requirement:** Compliant systems MAY implement this command.

801

802 **Usage:**

803 ReaderDevice.getFreeMemoryAlarmControl (void):

804 EdgeTriggeredAlarmControl

805

806 **Parameter(s):**

807 Data Type: void. This command takes no parameters.

808

809 **Return Value(s):**

810 Data Type: EdgeTriggeredAlarmControl. Reader will return its  
811 FreeMemoryAlarmControl object of type EdgeTriggeredAlarmControl.

812

813 **Possible Error Conditions:**

814 ERROR\_COMMAND\_NOT\_SUPPORTED

815 ERROR\_UNKNOWN

816

### 817 **5.1.11 ReaderDevice.getNTPServers**

818 The Host queries the Reader for a list of NTP servers used by the device to synchronize  
819 its current UTC clock (TimeUTC).

820

821 **Compliance Requirement:** Compliant systems MAY implement this command.

822

823 **Usage:**

824 `ReaderDevice.getNTPServers (void): string[]`

825

826 **Parameter(s):**

827 Data Type: `void`. This command takes no parameters.

828

829 **Return Value(s):**

830 Data Type: `string[]`. Reader will return a list of NTP servers used by the device to  
831 synchronize its current UTC clock (TimeUTC).

832

833 **Possible Error Conditions:**

834 `ERROR_COMMAND_NOT_SUPPORTED`

835 `ERROR_UNKNOWN`

836

### 837 **5.1.12 ReaderDevice.getDHCPServer**

838 The Host queries the Reader for the DHCP server currently used by the device for DHCP  
839 requests.

840

841 **Compliance Requirement:** Compliant systems MAY implement this command.

842

843 **Usage:**

844 `ReaderDevice.getDHCPServer (void): string`

845

846 **Parameter(s):**

847 Data Type: `void`. This command takes no parameters.

848

849 **Return Value(s):**

850 Data Type: `string`. Reader will return the DHCP server currently used by the device  
851 for DHCP requests.

852

853 **Possible Error Conditions:**

854 ERROR\_COMMAND\_NOT\_SUPPORTED

855 ERROR\_UNKNOWN

856

### 857 **5.1.13 ReaderDevice.getIOPort**

858 Returns the IOPort with the specified name currently associated with this Reader. If no  
859 IOPort object with the given name exists, the error ERROR\_IOPORT\_NOT\_FOUND  
860 is raised.

861

862 **Compliance Requirement:** Compliant systems SHALL implement this command.

863

864 **Usage:**

865 ReaderDevice.getIOPort (name: string): IOPort

866

867 **Parameter(s):**

868 name – Data Type: String. A name for the IOPort object which should be retrieved  
869 from the reader.

870

871 **Return Value(s):**

872 Data Type: IOPort. Reader will return the named IOPort object.

873

874 **Possible Error Conditions:**

875 ERROR\_COMMAND\_NOT\_SUPPORTED

876 ERROR\_IOPORT\_NOT\_FOUND

877 ERROR\_PARAMETER\_MISSING

878 ERROR\_PARAMETER\_INVALID\_DATATYPE

879 ERROR\_PARAMETER\_LENGTH\_EXCEEDED

880 ERROR\_UNKNOWN

### 881 **5.1.14 ReaderDevice.getAllIOPorts**

882 The Host queries the Reader for all its IOPort objects.

883

884 **Compliance Requirement:** Compliant systems SHALL implement this command.

885

886 **Usage:**

887 ReaderDevice.getAllIOPorts (void): IOPort[]

888

889 **Parameter(s):**

890 Data Type: void. This command takes no parameters.

891

892 **Return Value(s):**

893 Data Type: IOPort[] . Reader will return all its IOPort objects.

894

895 **Possible Error Conditions:**

896 ERROR\_COMMAND\_NOT\_SUPPORTED

897 ERROR\_UNKNOWN

898

899 **5.1.15 ReaderDevice.resetStatistics**

900 Instructs the reader to reset all statistical counters.

901 Upon receiving this command, the reader SHALL set all supported counters to zero.

902 Counters which are affected (if the reader supports them) are:

Object	Counters to reset
AntennaReadPoint	IdentificationCount, FailedIdentificationCount, MemReadCount, FailedMemReadCount, WriteCount, FailedWriteCount, KillCount, FailedKillCount, EraseCount, FailedEraseCount, LockCount, FailedLockCount, TimeEnergized
Source	UnknownToGlimpsedCount, GlimpsedToUnknownCount, GlimpsedToObservedCount, ObservedToLostCount, LostToGlimpsedCount, LostToUnknownCount
Trigger	FireCount

903

904 **Compliance Requirement:** Compliant systems SHALL implement this command.

905

906 **Usage:**

907 ReaderDevice.resetStatistics (void): void

908

909 **Parameter(s):**

910 Data Type: void. This command takes no parameters.

911

912 **Return Value(s):**

913 Data Type: void. This command will not return a value.

914

915 **Possible Error Conditions:**

916 ERROR\_COMMAND\_NOT\_SUPPORTED

917

### 918 **5.1.16 ReaderDevice.removeAlarmChannels**

919 Removes the specified AlarmChannels from the list of AlarmChannels currently  
920 associated with this Reader. If one or more of the AlarmChannels given are not  
921 known, or if some of the AlarmChannels to be removed are currently not associated  
922 with this Reader, these are ignored and all other AlarmChannels SHALL be removed  
923 and the command SHALL complete successfully.

924

925 **Compliance Requirement:** Compliant systems SHALL implement this command, if and  
926 only if, AlarmChannel.create() is implemented.

927

928 **Usage:**

929 ReaderDevice.removeAlarmChannels (channels:  
930 AlarmChannel[]): void

931

932 **Parameter(s):**

933 channels[] – Data Type: AlarmChannel. An array with AlarmChannel objects to  
934 be removed from the Reader.

935

936 **Return Value(s):**

937 Data Type: void. This command will not return a value.

938

939 **Possible Error Conditions:**

940 ERROR\_COMMAND\_NOT\_SUPPORTED

941 ERROR\_PARAMETER\_MISSING

942 ERROR\_PARAMETER\_INVALID\_DATATYPE

943 ERROR\_UNKNOWN

944

945 **5.1.17 ReaderDevice.removeAllAlarmChannels**

946 Removes all AlarmChannels currently associated with this Reader.

947

948 **Compliance Requirement:** Compliant systems SHALL implement this command if, and  
949 only if, AlarmChannel is implemented.

950

951 **Usage:**

952 ReaderDevice.removeAllAlarmChannels (void): void

953

954 **Parameter(s):**

955 Data Type: void. This command takes no parameters.

956

957 **Return Value(s):**

958 Data Type: void. This command will not return a value.

959

960 **Possible Error Conditions:**

961 ERROR\_COMMAND\_NOT\_SUPPORTED

962 ERROR\_UNKNOWN

963

964 **5.1.18 ReaderDevice.getAlarmChannel**

965 Returns the AlarmChannel with the specified name currently associated with this  
966 Reader. If no AlarmChannel object with the given name exists, the error  
967 ERROR\_ALARM\_CHANNEL\_NOT\_FOUND is raised.

968

969 **Compliance Requirement:** Compliant systems SHALL implement this command if, and  
970 only if, AlarmChannel.create() is implemented.

971

972 **Usage:**

973 ReaderDevice.getAlarmChannel (name: string): AlarmChannel

974

975 **Parameter(s):**

976 name – Data Type: String. A name for the AlarmChannel object which should be  
977 retrieved from the reader.

978



979 **Return Value(s):**

980 Data Type: AlarmChannel . Reader will return the named AlarmChannel object.

981

982 **Possible Error Conditions:**

983 ERROR\_COMMAND\_NOT\_SUPPORTED

984 ERROR\_ALARM\_CHANNEL\_NOT\_FOUND

985 ERROR\_PARAMETER\_MISSING

986 ERROR\_PARAMETER\_INVALID\_DATATYPE

987 ERROR\_PARAMETER\_LENGTH\_EXCEEDED

988 ERROR\_UNKNOWN

989

990 **5.1.19 ReaderDevice.getAllAlarmChannels**

991 Returns all AlarmChannels currently associated with the Reader. If no  
992 AlarmChannel are currently associated with this object, the command SHALL  
993 complete successfully and an empty list SHALL be returned.

994

995 **Compliance Requirement:** Compliant systems SHALL implement this command if, and  
996 only if, AlarmChannel.create() is implemented.

997

998 **Usage:**

999 ReaderDevice.getAllAlarmChannels (void): AlarmChannel[]

1000

1001 **Parameter(s):**

1002 Data Type: void . This command takes no parameters.

1003

1004 **Return Value(s):**

1005 Data Type: AlarmChannel[] . Reader will return an array with all defined  
1006 AlarmChannel objects for this Reader.

1007

1008 **Possible Error Conditions:**

1009 ERROR\_COMMAND\_NOT\_SUPPORTED

1010 ERROR\_UNKNOWN

1011

1012 **5.2 NotificationChannel**

1013

1014 **5.2.1 NotificationChannel.getLastNotificationAttempt**

1015 The Host queries the NotificationChannel object for the timestamp (TimeTicks) when the  
1016 last attempt was made to send a notification to the given address.

1017

1018 **Compliance Requirement:** Compliant systems MAY implement this command.

1019

1020 **Usage:**

1021 NotificationChannel.getLastNotificationAttempt (void):  
1022 timestamp

1023

1024 **Parameter(s):**

1025 Data Type: void. This command takes no parameters.

1026

1027 **Return Value(s):**

1028 Data Type: timestamp. NotificationChannel object will return the timestamp  
1029 (TimeTicks) when the last attempt was made to send a notification to the given address.

1030

1031 **Possible Error Conditions:**

1032 ERROR\_COMMAND\_NOT\_SUPPORTED

1033 ERROR\_UNKNOWN

1034 For a detailed error description see Reader Protocol chapter 8 Error Handling.

1035

1036

1037 **5.2.2 NotificationChannel.getLastSuccessfulNotification**

1038 The Host queries the NotificationChannel object for the timestamp (TimeTicks) when the  
1039 last successful notification was send to the given address.

1040

1041 **Compliance Requirement:** Compliant systems MAY implement this command.

1042

1043 **Usage:**

1044 NotificationChannel.getLastSuccessfulNotification (void):  
1045 timestamp

1046

1047 **Parameter(s):**

1048 Data Type: void. This command takes no parameters.

1049

1050 **Return Value(s):**

1051 Data Type: timestamp. NotificationChannel object will return the timestamp  
1052 (TimeTicks) when the last successful notification was send to the given address.

1053

1054 **Possible Error Conditions:**

1055 ERROR\_COMMAND\_NOT\_SUPPORTED

1056 ERROR\_UNKNOWN

1057

### 1058 **5.2.3 NotificationChannel.getOperStatus**

1059 The Host queries the NotificationChannel for its OperationalStatus attribute.

1060

1061 **Compliance Requirement:** Compliant systems SHALL implement this command if, and  
1062 only if, the system implements Notification Channels.

1063

1064 **Usage:**

1065 NotificationChannel.getOperStatus (void): OperationalStatus

1066

1067 **Parameter(s):**

1068 Data Type: void. This command takes no parameters.

1069

1070 **Return Value(s):**

1071 Data Type: OperationalStatus. NotificationChannel will return its  
1072 OperationalStatus attribute.

1073

1074 **Possible Error Conditions:**

1075 ERROR\_COMMAND\_NOT\_SUPPORTED

1076 ERROR\_UNKNOWN

1077

1078 **5.2.4 NotificationChannel.setAdminStatus**

1079 The Host sets the NotificationChannels AdminStatus object of type AdministrativeStatus.  
1080 The administrative state identifies if the NotificationChannel has been configured to be  
1081 operational. For example: UP or DOWN.

1082

1083 **Compliance Requirement:** Compliant systems SHALL implement this command if the  
1084 system implements Notification Channels.

1085

1086 **Usage:**

1087 NotificationChannel.setAdminStatus (administrativeStatus:  
1088 AdministrativeStatus): void

1089

1090 **Parameter(s):**

1091 administrativeStatus – Data Type: AdministrativeStatus. The  
1092 administrative state for the NotificationChannel.

1093

1094 **Return Value(s):**

1095 Data Type: void. This command will not return a value.

1096

1097 **Possible Error Conditions:**

1098

1099 ERROR\_PARAMETER\_MISSING

1100 ERROR\_PARAMETER\_INVALID\_DATATYPE

1101 ERROR\_UNKNOWN

1102

1103 **5.2.5 NotificationChannel.getAdminStatus**

1104 The Host queries the NotificationChannel for its AdminStatus object of type  
1105 AdministrativeStatus.

1106

1107 **Compliance Requirement:** Compliant systems SHALL implement this command if, and  
1108 only if, the system implements Notification Channels.

1109

1110 **Usage:**

1111 NotificationChannel.getAdminStatus (void):  
1112 AdministrativeStatus

1113

1114 **Parameter(s):**

1115 Data Type: void. This command takes no parameters.

1116

1117 **Return Value(s):**

1118 Data Type: AdministrativeStatus. NotificationChannel will return its

1119 AdminStatus object of type AdministrativeStatus.

1120

1121 **Possible Error Conditions:**

1122 ERROR\_COMMAND\_NOT\_SUPPORTED

1123 ERROR\_UNKNOWN

1124

1125 **5.2.6 NotificationChannel.getOperStatusAlarmControl**

1126 The Host queries the NotificationChannel object for its

1127 OperStatusAlarmControl attribute of type

1128 TTOperationalStatusAlarmControl.. This attribute is the object that controls

1129 the conditions for generating alarms alerting a manager of changes in a

1130 NotificationChannel's operational status.

1131

1132 **Compliance Requirement:** Compliant systems SHALL implement this command if, and

1133 only if, the system implements Notification Channels and AlarmChannels.

1134

1135 **Usage:**

1136 NotificationChannel.getOperStatusAlarmControl (void):

1137 TTOperationalStatusAlarmControl

1138

1139 **Parameter(s):**

1140 Data Type: void. This command takes no parameters.

1141

1142 **Return Value(s):**

1143 Data Type: TTOperationalStatusAlarmControl. The NotificationChannel

1144 will return its OperStatusAlarmControl object of type TTOperationalStatusAlarmControl.

1145

1146 **Possible Error Conditions:**

1147 ERROR\_COMMAND\_NOT\_SUPPORTED

1148 ERROR\_UNKNOWN

1149

## 1150 **5.3 AlarmChannel**

1151 NOTE: AlarmChannel objects are to be treated as RP Notification Channels, which  
1152 means that they use the same handshake and connection mechanisms. See [RP1] for  
1153 details.

### 1154 **5.3.1 AlarmChannel.create**

1155 Create an AlarmChannel object with a given name. If an AlarmChannel object  
1156 with the same name exists already, an error is returned. This is a static method.

1157

1158 The AlarmChannel SHALL implicitly be added to the list of all AlarmChannels  
1159 kept by the ReaderDevice object.

1160

1161 **Compliance Requirement:** Compliant systems MAY implement this command. If it is  
1162 not implemented, and Alarm Notifications are implemented, the system SHALL provide  
1163 an alternate method of defining AlarmChannels.

1164

#### 1165 **Usage:**

```
1166 static AlarmChannel.create(  
1167     name: string,  
1168     addr: address): AlarmChannel
```

1169

#### 1170 **Parameter(s):**

1171 name - Data Type: string . The name of the AlarmChannel to be created.

1172 addr - Data Type: address . The (host) address to which the reader will send alarms.

1173

#### 1174 **Return Value(s):**

1175 Data Type: AlarmChannel . The newly created object.

1176

#### 1177 **Possible Error Conditions:**

1178 ERROR\_COMMAND\_NOT\_SUPPORTED

1179 ERROR\_OBJECT\_EXISTS

1180 ERROR\_UNKNOWN

1181 ERROR\_PARAMETER\_MISSING

1182 ERROR\_PARAMETER\_INVALID\_DATATYPE  
1183 ERROR\_PARAMETER\_INVALID\_FORMAT  
1184 ERROR\_PARAMETER\_LENGTH\_EXCEEDED  
1185 ERROR\_AUTHORIZATION  
1186 ERROR\_TOO\_MANY\_ALARM\_CHANNELS  
1187

### 1188 **5.3.2 AlarmChannel.getName**

1189 Returns the name of the given AlarmChannel object.

1190

1191 **Compliance Requirement:** Compliant systems SHALL implement this command if  
1192 AlarmChannels are implemented.

1193

1194 **Usage:**

1195 AlarmChannel.getName(void): string

1196

1197 **Parameter(s):**

1198 Data Type: void. This command SHALL NOT take any parameters.

1199

1200 **Return Value(s):**

1201 Data Type: string. The name of the AlarmChannel object is returned.

1202

1203 **Possible Error Conditions:**

1204 ERROR\_COMMAND\_NOT\_SUPPORTED

1205 ERROR\_CHANNEL\_NOT\_FOUND

1206 ERROR\_UNKNOWN

1207

### 1208 **5.3.3 AlarmChannel.getAddress**

1209 Returns the (host) address to which this AlarmChannel object sends its alarms.

1210

1211 **Compliance Requirement:** Compliant systems SHALL implement this command if  
1212 AlarmChannels are implemented.

1213

1214 **Usage:**

1215 AlarmChannel.getAddress(void): address

1216

1217 **Parameter(s):**

1218 Data Type: void. This command SHALL NOT take any parameters.

1219

1220 **Return Value(s):**

1221 Data Type: address. The reporting address for this AlarmChannel.

1222

1223 **Possible Error Conditions:**

1224 ERROR\_CHANNEL\_NOT\_FOUND

1225 ERROR\_COMMAND\_NOT\_SUPPORTED

1226 ERROR\_UNKNOWN

1227

1228 **5.3.4 AlarmChannel.setAddress**

1229 Sets the (host) address to which this AlarmChannel object sends its alarms.

1230

1231 **Compliance Requirement:** Compliant systems SHALL implement this command if  
1232 AlarmChannel.create is implemented. Compliant systems MAY implement this  
1233 command if AlarmChannel is implemented by means of an alternate method without  
1234 implementing NotificationChannel.create..

1235

1236 **Usage:**

1237 AlarmChannel.setAddress(addr: address) : void

1238

1239 **Parameter(s):**

1240 addr - Data Type: address. The reporting address for this AlarmChannel.

1241

1242 **Return Value(s):**

1243 Data Type: void. This command SHALL NOT return any value.

1244

1245 **Possible Error Conditions:**

1246 ERROR\_COMMAND\_NOT\_SUPPORTED

1247 ERROR\_CHANNEL\_NOT\_FOUND



1248 ERROR\_UNKNOWN  
1249 ERROR\_PARAMETER\_MISSING  
1250 ERROR\_PARAMETER\_INVALID\_DATATYPE  
1251 ERROR\_PARAMETER\_INVALID\_FORMAT  
1252 ERROR\_PARAMETER\_LENGTH\_EXCEEDED  
1253 ERROR\_AUTHORIZATION  
1254

## 1255 **5.4 ReadPoint**

1256  
1257  
1258  
1259  
1260  
1261  
1262

### 1263 **5.4.1 ReadPoint.getClassName**

1264 This method returns the class name of the ReadPoint object. In the current specification,  
1265 the only supported read point class is an "AntennaReadPoint", but other types may be  
1266 introduced in future versions.

1267

1268 **Compliance Requirement:** Compliant systems SHALL implement this command.

1269 **Compliance Requirement:** Compliant systems SHALL implement this command.

1270

1271 **Usage:**

1272 `ReadPoint.getClassName(void): string`

1273

1274 **Parameter(s):**

1275 Data Type: void. This command takes no parameters.

1276

1277 **Return Value(s):**

1278 Data Type: string. The class name for the ReadPoint. The only supported return value is  
1279 "AntennaReadPoint". Note that future versions of the specification may support other  
1280 return values.

1281

1282 **Possible Error Conditions:**

1283 ERROR\_UNKNOWN

### 1284 **5.4.2 ReadPoint.getDescription**

1285 Returns a description of this ReadPoint.

1286 The description is set via a call to setDescription and returned via this call. In the event  
1287 there has been no previous call to setDescription then the returned string will have 0  
1288 length.

1289

1290 **Compliance Requirement:** Compliant systems SHALL implement this command .

1291

1292 **Usage:**

1293 ReadPoint.getDescription (void): string

1294

1295 **Parameter(s):**

1296 Data Type: void. This command takes no parameters.

1297

1298 **Return Value(s):**

1299 Data Type: string. The readpoint descriptive name.

1300

1301 **Possible Error Conditions:**

1302 ERROR\_UNKNOWN

1303

### 1304 **5.4.3 ReadPoint.setDescription**

1305 Set the current ReadPoint description.

1306 This may later be retrieved using ReadPoint.getDescription.

1307

1308 **Compliance Requirement:** Compliant systems SHALL implement this command.

1309 **Usage:**

1310 ReadPoint.setDescription(description: string): void

1311

1312 **Parameter(s):**

1313 description – Data Type: String. A description for this ReadPoint. The  
1314 maximum allowable length for this string is vendor-dependent. Attempts to set a string  
1315 of excessive length will result in a “length exceeded” error.

1316 **Return Value(s):**

1317 Data Type: void. This command will not return a value.

1318

1319 **Possible Error Conditions:**

1320 ERROR\_PARAMETER\_MISSING

1321 ERROR\_PARAMETER\_INVALID\_DATATYPE

1322 ERROR\_PARAMETER\_INVALID\_VALUE

1323 ERROR\_PARAMETER\_LENGTH\_EXCEEDED

1324 ERROR\_UNKNOWN

1325 ERROR\_AUTHORIZATION

1326

1327 **5.4.4 ReadPoint.getAdminStatus**

1328 Returns the current ReadPoint administrative status.

1329 The administrative status represents the host’s *desired* status for this ReadPoint. This  
1330 differs from the operational status which represents the *actual* ReadPoint status. The  
1331 administrative status is set via a call to setAdminStatus and getAdminStatus simply  
1332 returns the most-recently-set value, or the initial vendor-specific administrative state if  
1333 setAdminStatus has not yet been called.

1334 **Compliance Requirement:** Compliant systems SHALL implement this command.

1335 **Usage:**

1336 ReadPoint.getAdminStatus (void): AdministrativeStatus

1337

1338 **Parameter(s):**

1339 Data Type: void. This command takes no parameters.

1340

1341 **Return Value(s):**

1342 Data Type: AdministrativeStatus.

1343

1344 **Possible Error Conditions:**

1345 ERROR\_UNKNOWN

1346 For a detailed description of these error codes see the Reader Protocol Specification 1.1.

1347

## 1348 **5.4.5 ReadPoint.setAdminStatus**

1349 Set the current ReadPoint administrative status.

1350 The administrative status represents the host's *desired* status for this ReadPoint. This  
1351 differs from the operational status which represents the *actual* ReadPoint status. Hosts  
1352 may query the most-recently-requested status using `getAdminStatus`.

1353

1354 **Compliance Requirement:** Compliant systems SHALL implement this command.

1355 **Usage:**

```
1356 ReadPoint.setAdminStatus (status: AdministrativeStatus):  
1357 void
```

1358

1359 **Parameter(s):**

1360 `status` - Data Type: `AdministrativeStatus`. The desired  
1361 administrative status of the ReadPoint.

1362

1363 **Return Value(s):**

1364 Data Type: `void`. This command will not return a value.

1365

1366 **Possible Error Conditions:**

1367 `ERROR_PARAMETER_MISSING`

1368 `ERROR_PARAMETER_INVALID_DATATYPE`

1369 `ERROR_PARAMETER_INVALID_VALUE`

1370 `ERROR_UNKNOWN`

1371

## 1372 **5.4.6 ReadPoint.getOperStatus**

1373 Returns the ReadPoint's current operational status.

1374 The operational status represents the actual status of the ReadPoint. It may be polled  
1375 using `getOperStatus` or monitored via alarms (see `GetOperStatusAlarmControl`).

1376

1377 **Compliance Requirement:** Compliant systems SHALL implement this command.

1378 **Usage:**

```
1379 ReadPoint.getOperStatus (void): OperationalStatus
```

1380

1381 **Parameter(s):**

1382 Data Type: void. This command takes no parameters.

1383

1384 **Return Value(s):**

1385 Data Type: OperationalStatus. The operational status of the  
1386 ReadPoint. It may be "unknown", "up", "down" or "other".  
1387 See the definition of the OperationalStatus enumeration for  
1388 details.

1389

1390 **Possible Error Conditions:**

1391 ERROR\_UNKNOWN

1392

1393 **5.4.7 ReadPoint.getOperStatusAlarmControl**

1394 The Host queries the ReadPoint object for its OperStatusAlarmControl  
1395 attribute of type TOperationalStatusAlarmControl.. This attribute is the  
1396 object that controls the conditions for generating alarms alerting a manager of changes in  
1397 a ReadPoint's operational status.

1398 In addition to an alarm, the operational status may also be polled (see getOperStatus).

1399

1400 **Compliance Requirement:** Compliant systems SHALL implement this command if, and  
1401 only if, AlarmChannels are implemented.

1402

1403 **Usage:**

1404 ReadPoint.getOperStatusAlarmControl (void):  
1405 TOperationalStatusAlarmControl

1406

1407 **Parameter(s):**

1408 Data Type: void. This command takes no parameters.

1409

1410 **Return Value(s):**

1411 Data Type: TOperationalStatusAlarmControl. An alarm control  
1412 for monitoring the operational status of the ReadPoint.

1413

1414 **Possible Error Conditions:**

1415 ERROR\_COMMAND\_NOT\_SUPPORTED

1416 ERROR\_UNKNOWN

1417

## 1418 **5.5 AntennaReadPoint**

1419 AntennaReadPoint extends ReadPoint. It is used to track details about the Radio  
1420 Frequency characteristics and RFID statistics for the reader device.

### 1421 **5.5.1 AntennaReadPoint.getIdentificationCount**

1422 Returns the number of the successful tags that have been identified across an  
1423 AntennaReadPoint. In other words, the number of tags whose unique identifier has  
1424 been successfully detected by the AntennaReadPoint.

1425 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1426 manually reset via a call to ReaderDevice.resetStatistics.

1427 Note that if a single tag's identifier is read multiple times, then the read count will  
1428 increment on every successful read. It is the number of times *any* tag's identifier is read,  
1429 not the number of *unique* tags which have been read.

1430

1431 **Compliance Requirement:** Compliant systems SHALL implement this command.

1432

1433 **Usage:**

1434 AntennaReadPoint.getIdentificationCount (void): integer

1435

1436 **Parameter(s):**

1437 Data Type: void. This command takes no parameters.

1438

1439 **Return Value(s):**

1440 Data Type: integer. The count of the successful tag identifiers read at this  
1441 AntennaReadPoint.

1442

1443 **Possible Error Conditions:**

1444 ERROR\_UNKNOWN

1445

### 1446 **5.5.2 AntennaReadPoint.getFailedIdentificationCount**

1447 Returns the number of the failed tag identification attempts at the  
1448 AntennaReadPoint.

1449 This count only includes failures where the reader can unambiguously determine that a  
1450 tag was in field that could not be identified. If the reader does not have the means to  
1451 determine this, this counter may be left at 0.

1452 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1453 manually reset via a call to `ReaderDevice.resetStatistics`.

1454

1455 **Compliance Requirement:** Compliant systems MAY implement this command.

1456

1457 **Usage:**

1458 `AntennaReadPoint.getFailedIdentificationCount (void):`  
1459 `integer`

1460

1461 **Parameter(s):**

1462 Data Type: `void`. This command takes no parameters.

1463

1464 **Return Value(s):**

1465 Data Type: `integer`. The count of the failed attempts at reading the identifier for a  
1466 tag at this antenna `ReadPoint`.

1467

1468 **Possible Error Conditions:**

1469 `ERROR_COMMAND_NOT_SUPPORTED`

1470 `ERROR_UNKNOWN`

1471

### 1472 **5.5.3 AntennaReadPoint.getMemReadCount**

1473 Returns the number of tag memory reads at the `AntennaReadPoint`. This should not  
1474 be confused with the number of times the tag has been successfully queried or has  
1475 communicated its unique identifier. Such statistics is reported by  
1476 `AntennaReadPoint.getIdentificationCount`.

1477 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1478 manually reset via a call to `ReaderDevice.resetStatistics`.

1479

1480 **Compliance Requirement:** Compliant systems MAY implement this command.

1481

1482 **Usage:**

1483 `AntennaReadPoint.getMemReadCount (void): integer`

1484

1485 **Parameter(s):**

1486 Data Type: void. This command takes no parameters.

1487

1488 **Return Value(s):**

1489 Data Type: integer. The count of the successful tag memory reads at this antenna  
1490 ReadPoint.

1491

1492 **Possible Error Conditions:**

1493 ERROR\_COMMAND\_NOT\_SUPPORTED

1494 ERROR\_UNKNOWN

1495 **5.5.4 AntennaReadPoint.getFailedMemReadCount**

1496 Returns the number of the failed tag memory reads at the AntennaReadPoint.

1497 This count only includes failures where it attempts to read tag memory but does not  
1498 complete successfully.

1499 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1500 manually reset via a call to ReaderDevice.resetStatistics.

1501

1502 **Compliance Requirement:** Compliant systems MAY implement this command.

1503

1504 **Usage:**

1505 AntennaReadPoint.getFailedMemReadCount (void): integer

1506

1507 **Parameter(s):**

1508 Data Type: void. This command takes no parameters.

1509

1510 **Return Value(s):**

1511 Data Type: integer. The count of the failed tag memory reads at this antenna  
1512 readpoint.

1513

1514 **Possible Error Conditions:**

1515 ERROR\_COMMAND\_NOT\_SUPPORTED

1516 ERROR\_UNKNOWN



1517 **5.5.5 AntennaReadPoint.getFailedMemReadAlarmControl**

1518 The Host queries the `AntennaReadPoint` object for its  
1519 `FailedMemReadAlarmControl` attribute of type `AlarmControl`. This alarm  
1520 control object controls the generation of alarms triggered by failed read operations of  
1521 memory banks across the `AntennaRead` point.

1522 In addition to an alarm, the status of failed memory reads may also be polled (see  
1523 `getMemReadCount`).

1524

1525 **Compliance Requirement:** Compliant systems MAY implement this command.

1526

1527 **Usage:**

1528 `AntennaReadPoint.getFailedMemReadAlarmControl`  
1529 `(void):AlarmControl`

1530

1531 **Parameter(s):**

1532 Data Type: `void`. This command takes no parameters.

1533

1534 **Return Value(s):**

1535 Data Type: `AlarmControl`. An alarm control for monitoring tag memory read  
1536 failures.

1537

1538 **Possible Error Conditions:**

1539 `ERROR_COMMAND_NOT_SUPPORTED`

1540 `ERROR_UNKNOWN`

1541 **5.5.6 AntennaReadPoint.getWriteCount**

1542 Returns the number of successful tag writes at the `AntennaReadPoint`, including  
1543 writes to any of the memory banks including the tag identifier.

1544 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1545 manually reset via a call to `ReaderDevice.resetStatistics`.

1546 Note that if a single tag is written multiple times, then the count will increment on every  
1547 successful write. It is the number of times *any* tag has been written, not the number of  
1548 *unique* tags which have been written.

1549 **Compliance Requirement:** Compliant systems SHALL implement this command.

1550

1551 **Usage:**

1552 `AntennaReadPoint.getWriteCount (void): integer`

1553

1554 **Parameter(s):**

1555 Data Type: `void`. This command takes no parameters.

1556

1557 **Return Value(s):**

1558 Data Type: `integer`. The count of the successful writes at this

1559 `AntennaReadPoint`. If Write functionality is not supported by the Reader it must

1560 return 0 every time.

1561

1562 **Possible Error Conditions:**

1563 `ERROR_UNKNOWN`

1564 For a detailed description of these error codes see the Reader Protocol Specification 1.1.

1565

### 1566 **5.5.7 AntennaReadPoint.getFailedWriteCount**

1567 Returns the number of the failed attempts to write tags at the `AntennaReadPoint`,  
1568 including failed writes to any of the memory banks including the tag identifier.

1569 This is the number of times the reader attempted to write a tag, but the operation failed to  
1570 complete successfully.

1571 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1572 manually reset via a call to `ReaderDevice.resetStatistics`.

1573

1574 **Compliance Requirement:** Compliant systems MAY implement this command.

1575

1576 **Usage:**

1577 `AntennaReadPoint.getFailedWriteCount (void): integer`

1578

1579 **Parameter(s):**

1580 Data Type: `void`. This command takes no parameters.

1581

1582 **Return Value(s):**

1583 Data Type: `integer`. The count of the failed writes at this `AntennaReadPoint`.

1584

1585 **Possible Error Conditions:**

1586 ERROR\_COMMAND\_NOT\_SUPPORTED

1587 ERROR\_UNKNOWN

1588

### 1589 **5.5.8 AntennaReadPoint.getFailedWriteAlarmControl**

1590 The Host queries the `AntennaReadPoint` object for its  
1591 `FailedWriteAlarmControl` attribute of type `AlarmControl`. This alarm control  
1592 object controls the generation of alarms triggered by failed write operations across the  
1593 `AntennaRead` point, including failed writes to any of the memory banks including the tag  
1594 identifier.

1595 In addition to an alarm, the status of writes may also be polled (see `getWriteCount` and  
1596 `getFailedWriteCount`).

1597

1598 **Compliance Requirement:** Compliant systems MAY implement this command.

1599

#### 1600 **Usage:**

1601 `AntennaReadPoint.getFailedWriteAlarmControl`  
1602 `(void):AlarmControl`

1603

#### 1604 **Parameter(s):**

1605 Data Type: `void`. This command takes no parameters.

1606

#### 1607 **Return Value(s):**

1608 Data Type: `AlarmControl`. An alarm control for monitoring the number of failed  
1609 writes.

1610

#### 1611 **Possible Error Conditions:**

1612 ERROR\_COMMAND\_NOT\_SUPPORTED

1613 ERROR\_UNKNOWN

1614 For a detailed description of these error codes see the Reader Protocol Specification 1.1.

### 1615 **5.5.9 AntennaReadPoint.getKillCount**

1616 Returns the number of tags successfully killed at the `AntennaReadPoint`.

1617 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1618 manually reset via a call to `ReaderDevice.resetStatistics`.

1619

1620 **Compliance Requirement:** Compliant systems SHALL implement this command.

1621

1622 **Usage:**

1623 `AntennaReadPoint.getKillCount (void): integer`

1624

1625 **Parameter(s):**

1626 Data Type: void. This command takes no parameters.

1627

1628 **Return Value(s):**

1629 Data Type: integer . The count of the successful tag kills at this antenna ReadPoint.

1630

1631 **Possible Error Conditions:**

1632 ERROR\_UNKNOWN

1633

#### 1634 **5.5.10 AntennaReadPoint.getFailedKillCount**

1635 Returns the number of the failed tag kills at the AntennaReadPoint.

1636 This count only includes failures where it attempts to kill a tag but the kill does not  
1637 complete successfully.

1638 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1639 manually reset via a call to `ReaderDevice.resetStatistics`.

1640

1641 **Compliance Requirement:** Compliant systems MAY implement this command.

1642

1643 **Usage:**

1644 `AntennaReadPoint.getFailedKillCount (void): integer`

1645

1646 **Parameter(s):**

1647 Data Type: void. This command takes no parameters.

1648

1649 **Return Value(s):**

1650 Data Type: integer . The count of the failed tag kills at this antenna readpoint.

1651

1652 **Possible Error Conditions:**

1653 ERROR\_COMMAND\_NOT\_SUPPORTED

1654 ERROR\_UNKNOWN

### 1655 **5.5.11 AntennaReadPoint.getFailedKillAlarmControl**

1656 Return the AntennaReadPoint's failed kill alarm control.

1657 In addition to an alarm, the status of kills may also be polled (see getKillCount and  
1658 getFailedKillCount).

1659

1660 **Compliance Requirement:** Compliant systems MAY implement this command.

1661

1662 **Usage:**

1663 `AntennaReadPoint.getFailedKillAlarmControl`

1664 `(void):AlarmControl`

1665

1666 **Parameter(s):**

1667 Data Type: void. This command takes no parameters.

1668

1669 **Return Value(s):**

1670 Data Type: AlarmControl. An alarm control for monitoring tag kill failures.

1671

1672 **Possible Error Conditions:**

1673 ERROR\_COMMAND\_NOT\_SUPPORTED

1674 ERROR\_UNKNOWN

### 1675 **5.5.12 AntennaReadPoint.getEraseCount**

1676 Returns the number of tags successfully erased at the AntennaReadPoint.

1677 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1678 manually reset via a call to ReaderDevice.resetStatistics.

1679

1680 **Compliance Requirement:** Compliant systems SHALL implement this command.

1681

1682 **Usage:**

1683 `AntennaReadPoint.getEraseCount (void): integer`

1684

1685 **Parameter(s):**

1686 Data Type: void. This command takes no parameters.

1687

1688 **Return Value(s):**

1689 Data Type: integer. The count of the successful tag erasures at this antenna

1690 ReadPoint.

1691

1692 **Possible Error Conditions:**

1693 ERROR\_UNKNOWN

1694

1695 **5.5.13 AntennaReadPoint.getFailedEraseCount**

1696 Returns the number of the failed tag erasures at the AntennaReadPoint.

1697 This count only includes failures where it attempts to erase a tag but the erasure does not  
1698 complete successfully.

1699 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1700 manually reset via a call to ReaderDevice.resetStatistics.

1701

1702 **Compliance Requirement:** Compliant systems MAY implement this command.

1703

1704 **Usage:**

1705 AntennaReadPoint.getFailedEraseCount (void): integer

1706

1707 **Parameter(s):**

1708 Data Type: void. This command takes no parameters.

1709

1710 **Return Value(s):**

1711 Data Type: integer. The count of the failed tag erasures at this antenna readpoint.

1712

1713 **Possible Error Conditions:**

1714 ERROR\_COMMAND\_NOT\_SUPPORTED

1715 ERROR\_UNKNOWN

1716 **5.5.14 AntennaReadPoint.getFailedEraseAlarmControl**

1717 Return the AntennaReadPoint's failed erase alarm control.

1718 In addition to an alarm, the status of erasures may also be polled (see `getEraseCount` and  
1719 `getFailedEraseCount`).

1720

1721 **Compliance Requirement:** Compliant systems MAY implement this command.

1722

1723 **Usage:**

1724 `AntennaReadPoint.getFailedEraseAlarmControl`  
1725 `(void):AlarmControl`

1726

1727 **Parameter(s):**

1728 Data Type: `void`. This command takes no parameters.

1729

1730 **Return Value(s):**

1731 Data Type: `AlarmControl`. An alarm control for monitoring tag erasure failures.

1732

1733 **Possible Error Conditions:**

1734 `ERROR_COMMAND_NOT_SUPPORTED`

1735 `ERROR_UNKNOWN`

### 1736 **5.5.15 AntennaReadPoint.getLockCount**

1737 Returns the number of tags successfully locked at the `AntennaReadPoint`.

1738 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1739 manually reset via a call to `ReaderDevice.resetStatistics`.

1740

1741 **Compliance Requirement:** Compliant systems MAY implement this command.

1742

1743 **Usage:**

1744 `AntennaReadPoint.getLockCount (void): integer`

1745

1746 **Parameter(s):**

1747 Data Type: `void`. This command takes no parameters.

1748

1749 **Return Value(s):**

1750 Data Type: `integer`. The count of tags successfully locked at this antenna `ReadPoint`.

1751

1752 **Possible Error Conditions:**

1753 ERROR\_COMMAND\_NOT\_SUPPORTED

1754 ERROR\_UNKNOWN

### 1755 **5.5.16 AntennaReadPoint.getFailedLockCount**

1756 Returns the number of the failed tag locks at the AntennaReadPoint.

1757 This count only includes failures where it attempts to lock a tag but the lock does not  
1758 complete successfully.

1759 This count is automatically reset to 0 whenever the reader is restarted, and may be  
1760 manually reset via a call to ReaderDevice.resetStatistics.

1761

1762 **Compliance Requirement:** Compliant systems MAY implement this command.

1763

1764 **Usage:**

1765 AntennaReadPoint.getFailedLockCount (void): integer

1766

1767 **Parameter(s):**

1768 Data Type: void. This command takes no parameters.

1769

1770 **Return Value(s):**

1771 Data Type: integer. The count of the failed tag locks at this antenna readpoint.

1772

1773 **Possible Error Conditions:**

1774 ERROR\_COMMAND\_NOT\_SUPPORTED

1775 ERROR\_UNKNOWN

1776 AntennaReadPoint.getFailedLockAlarmControl

1777

1778

1779

1780

1781

1782

1783



1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801

1802 **5.5.16.1**

1803 Return the AntennaReadPoint's failed lock alarm control.

1804 In addition to an alarm, the status of locks may also be polled (see getLockCount and  
1805 getFailedLockCount).

1806

1807 **Compliance Requirement:** Compliant systems MAY implement this command.

1808

1809 **Usage:**

1810 `AntennaReadPoint.getFailedLockAlarmControl`  
1811 `(void):AlarmControl`

1812

1813 **Parameter(s):**

1814 Data Type: void. This command takes no parameters.

1815

1816 **Return Value(s):**

1817 Data Type: AlarmControl. An alarm control for monitoring tag lock failures.

1818

1819 **Possible Error Conditions:**

1820 ERROR\_COMMAND\_NOT\_SUPPORTED

1821 ERROR\_UNKNOWN

### 1822 **5.5.17 AntennaReadPoint.getTimeEnergized**

1823 Returns the number of milliseconds the AntennaReadPoint has been  
1824 energized in order to communicate with tags.

1825 **Compliance Requirement:** Compliant systems MAY implement this command.

1826

1827 **Usage:**

1828 AntennaReadPoint.getTimeEnergized(void): integer

1829

1830 **Parameter(s):**

1831 Data Type: void. This command takes no parameters.

1832

1833 **Return Value(s):**

1834 Data Type: integer. The number of milliseconds the AntennaReadPoint has  
1835 been energized attempting communication with tags. This count is  
1836 automatically reset to 0 whenever the reader is restarted, and may be manually reset via a  
1837 call to ReaderDevice.resetStatistics.

1838

1839 **Possible Error Conditions:**

1840 ERROR\_COMMAND\_NOT\_SUPPORTED

1841 ERROR\_UNKNOWN

1842 AntennaReadPoint.getPowerLevel

1843

1844

1845

1846

1847

1848

1849

1850

1851 **5.5.17.1**

1852 Returns the current transmit power level of the `AntennaReadPoint`. The units of  
1853 measurement are vendor specific.

1854

1855 **Compliance Requirement:** Compliant systems MAY implement this command.

1856

1857 **Usage:**

1858 `AntennaReadPoint.getPowerLevel(void): integer`

1859

1860 **Parameter(s):**

1861 Data Type: `void`. This command takes no parameters.

1862

1863 **Return Value(s):**

1864 Data Type: `integer`. The current transmit power level of the `AntennaReadPoint`.

1865 The meaning of this value is device-dependent and users should consult documentation

1866 provided by the reader manufacturer.

1867

1868 **Possible Error Conditions:**

1869 `ERROR_COMMAND_NOT_SUPPORTED`

1870 `ERROR_UNKNOWN`

1871

1872 **5.5.18      `AntennaReadPoint.getNoiseLevel`**

1873 The Host sends a `getNoiseLevel` message to query the Reader for the current noise

1874 level observed at the `AntennaReadPoint`. The units of measurement are vendor

1875 specific.

1876

1877 **Compliance Requirement:** Compliant systems MAY implement this command.

1878

1879 **Usage:**

1880 `AntennaReadPoint.getNoiseLevel(void): integer`

1881

1882 **Parameter(s):**

1883 Data Type: `void`. This command takes no parameters.

1884

1885 **Return Value(s):**

1886 Data Type: `integer`. The current noise level observed at the `AntennaReadPoint`.  
1887 The meaning of this value is device-dependent and users should consult the  
1888 documentation provided by the reader manufacturer for the meaning. The only constraint  
1889 is that larger numbers imply higher levels of noise.

1890

1891 **Possible Error Conditions:**

1892 `ERROR_COMMAND_NOT_SUPPORTED`

1893 `ERROR_UNKNOWN`

1894

1895 **5.6 Source Object**

1896 **5.6.1 Source.getUnknownToGlimpsedCount**

1897 The Host queries the Reader for the number of times a transition from state `Unknown` to  
1898 state `Glimpsed` have been detected for the particular source.

1899

1900 **Compliance Requirement:** Compliant systems **MAY** implement this command.

1901

1902 **Usage:**

1903 `Source.getUnknownToGlimpsedCount (void): integer`

1904

1905 **Parameter(s):**

1906 Data Type: `void`. This command takes no parameters.

1907

1908 **Return Value(s):**

1909 Data Type: `integer`. The number times the particular transition has been detected.

1910

1911 **Possible Error Conditions:**

1912 `ERROR_SOURCE_NOT_FOUND`

1913 `ERROR_COMMAND_NOT_SUPPORTED`

1914

1915 **5.6.2 Source.getGlimpsedToUnknownCount**

1916 The Host queries the Reader for the number of times a transition from state `Glimpsed` to  
1917 state `Unknown` have been detected for the particular source.

1918

1919 **Compliance Requirement:** Compliant systems MAY implement this command.

1920

1921 **Usage:**

1922 `Source.getGlimpsedToUnknownCount (void): integer`

1923

1924 **Parameter(s):**

1925 Data Type: `void`. This command takes no parameters.

1926

1927 **Return Value(s):**

1928 Data Type: `integer`. The number times the particular transition has been detected.

1929

1930 **Possible Error Conditions:**

1931 `ERROR_SOURCE_NOT_FOUND`

1932 `ERROR_COMMAND_NOT_SUPPORTED`

1933

### 1934 **5.6.3 Source.getGlimpsedToObservedCount**

1935 The Host queries the Reader for the number of times a transition from state Glimpsed to  
1936 state Observed have been detected for the particular source.

1937

1938 **Compliance Requirement:** Compliant systems MAY implement this command.

1939

1940 **Usage:**

1941 `Source.getGlimpsedToObservedCount (void): integer`

1942

1943 **Parameter(s):**

1944 Data Type: `void`. This command takes no parameters.

1945

1946 **Return Value(s):**

1947 Data Type: `integer`. The number times the particular transition has been detected.

1948

1949 **Possible Error Conditions:**

1950 `ERROR_SOURCE_NOT_FOUND`

1951 ERROR\_COMMAND\_NOT\_SUPPORTED

1952

#### 1953 **5.6.4 Source.getObservedToLostCount**

1954 The Host queries the Reader for the number of times a transition from state Observed to  
1955 state Lost have been detected for the particular source.

1956

1957 **Compliance Requirement:** Compliant systems MAY implement this command.

1958

1959 **Usage:**

1960 `Source.getObservedToLostCount (void): integer`

1961

1962 **Parameter(s):**

1963 Data Type: `void`. This command takes no parameters.

1964

1965 **Return Value(s):**

1966 Data Type: `integer`. The number times the particular transition has been detected.

1967

1968 **Possible Error Conditions:**

1969 ERROR\_SOURCE\_NOT\_FOUND

1970 ERROR\_COMMAND\_NOT\_SUPPORTED

1971

#### 1972 **5.6.5 Source.getLostToGlimpsedCount**

1973 The Host queries the Reader for the number of times a transition from state Lost to state  
1974 Observed have been detected for the particular source.

1975

1976 **Compliance Requirement:** Compliant systems MAY implement this command.

1977

1978 **Usage:**

1979 `Source.getLostToGlimpsedCount (void): integer`

1980

1981 **Parameter(s):**

1982 Data Type: `void`. This command takes no parameters.

1983

1984 **Return Value(s):**

1985 Data Type: `integer`. The number times the particular transition has been detected.

1986

1987 **Possible Error Conditions:**

1988 `ERROR_SOURCE_NOT_FOUND`

1989 `ERROR_COMMAND_NOT_SUPPORTED`

1990

### 1991 **5.6.6 Source.getLostToUnknownCount**

1992 The Host queries the Reader for the number of times a transition from state Lost to state  
1993 Unknown have been detected for the particular source.

1994

1995 **Compliance Requirement:** Compliant systems MAY implement this command.

1996

1997 **Usage:**

1998 `Source.getLostToUnknownCount (void): integer`

1999

2000 **Parameter(s):**

2001 Data Type: `void`. This command takes no parameters.

2002

2003 **Return Value(s):**

2004 Data Type: `integer`. The number times the particular transition has been detected.

2005

2006 **Possible Error Conditions:**

2007 `ERROR_SOURCE_NOT_FOUND`

2008 `ERROR_COMMAND_NOT_SUPPORTED`

2009

### 2010 **5.6.7 Source.getOperStatus**

2011 The Host queries the Reader for the operational status of this particular Source. The  
2012 operational status reported is one of the values in the `OperationalStatus` enumeration, e.g.  
2013 `UP`, `DOWN`, `UNKNOWN` or `OTHER`. The operational status represents the actual status  
2014 of the Source. It may be polled using `getOperStatus` or monitored via alarms (see  
2015 `GetOperStatusAlarmControl`).

2016

2017

2018 **Compliance Requirement:** Compliant systems SHALL implement this command.

2019 **Usage:**

2020 `Source.getOperStatus (void): OperationalStatus`

2021

2022 **Parameter(s):**

2023 Data Type: `void`. This command takes no parameters.

2024

2025 **Return Value(s):**

2026 Data Type: `OperationalStatus`. Reader will return the current operational status of  
2027 this Source.

2028

2029 **Possible Error Conditions:**

2030 `ERROR_COMMAND_NOT_SUPPORTED`

2031 `ERROR_SOURCE_NOT_FOUND`

2032

### 2033 **5.6.8 Source.getAdminStatus**

2034 The Host queries the Reader for the administrative status of a particular Source. The  
2035 administrative status reported is one of the values in the `AdministrativeStatus`  
2036 enumeration, e.g. UP or DOWN.

2037 The administrative status represents the host's *desired* status for this Source. This differs  
2038 from the operational status which represents the *actual* Source status. The administrative  
2039 status is set via a call to `setAdminStaus` and `getAdminStatus` simply returns the most-  
2040 recently-set value.

2041 **Compliance Requirement:** Compliant systems SHALL implement this command.

2042 **Usage:**

2043 `Source.getAdminStatus (void): AdministrativeStatus`

2044

2045 **Parameter(s):**

2046 Data Type: `void`. This command takes no parameters.

2047

2048 **Return Value(s):**

2049 Data Type: `AdministrativeStatus`. Reader will return the current administrative  
2050 status of this Source.



2051

2052 **Possible Error Conditions:**

2053 ERROR\_SOURCE\_NOT\_FOUND

2054 ERROR\_COMMAND\_NOT\_SUPPORTED

2055

### 2056 **5.6.9 Source.setAdminStatus**

2057 This operation is used to set the administrative status of a particular Source. The  
2058 administrative status assigned is one of the values in the AdministrativeStatus  
2059 enumeration, e.g. UP or DOWN

2060 The administrative status represents the host's *desired* status for this Source. This differs  
2061 from the operational status which represents the *actual* status. Hosts may query the most-  
2062 recently-requested status using getAdminStatus.

2063

2064 **Compliance Requirement:** Compliant systems SHALL implement this command.

2065

2066 **Usage:**

2067 Source.setAdminStatus (administrativeStatus  
2068 :AdministrativeStatus): void

2069

2070 **Parameter(s):**

2071 administrativeStatus – Data Type: AdministrativeStatus. This is the  
2072 new administrative status to be set.

2073

2074 **Return Value(s):**

2075 Data Type: void. This command will not return a value.

2076

2077 **Possible Error Conditions:**

2078 ERROR\_COMMAND\_NOT\_SUPPORTED

2079 ERROR\_INVALID\_ADMIN\_STATUS

2080

### 2081 **5.6.10 Source.getOperStatusAlarmControl**

2082 The Host queries the Source object for its OperStatusAlarmControl attribute of  
2083 type TOperationalStatusAlarmControl. This attribute is the object that  
2084 controls the conditions for generating alarms alerting a manager of changes in a  
2085 Source's operational status.

2086 In addition to an alarm, the operational status may also be polled (see `getOperStatus`).

2087

2088 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2089 `AlarmChannel` is implemented.

2090

2091 **Usage:**

2092 `Source.getOperStatusAlarm (void):`

2093 `TTOperationalStatusAlarmControl`

2094

2095 **Parameter(s):**

2096 Data Type: `void`. This command takes no parameters.

2097

2098 **Return Value(s):**

2099 Data Type: `TTOperationalStatusAlarmControl`. An alarm control for  
2100 monitoring the operational status of the Source.

2101

2102 **Possible Error Conditions:**

2103 `ERROR_COMMAND_NOT_FOUND`

2104 `ERROR_SOURCE_NOT_FOUND`

2105

## 2106 **5.7 Trigger Object**

### 2107 **5.7.1 Trigger.getFireCount**

2108 The Host queries the Reader for the number of times a particular trigger has fired.

2109

2110 **Compliance Requirement:** Compliant systems MAY implement this command.

2111

2112 **Usage:**

2113 `Trigger.getFireCount (void): integer`

2114

2115 **Parameter(s):**

2116 Data Type: `void`. This command takes no parameters.

2117

2118 **Return Value(s):**

2119 Data Type: `integer` . Reader will return the number of times the trigger has fired.

2120

2121 **Possible Error Conditions:**

2122 `ERROR_TRIGGER_NOT_FOUND`

2123 `ERROR_COMMAND_NOT_SUPPORTED`

2124

## 2125 **5.8 IOPort Object**

### 2126 **5.8.1 IOPort.getName**

2127 The Host queries the Reader for the name of a particular IO-port. Since the IO-port  
2128 names are set by the manufacturer, there is no `setName ()` command.

2129

2130 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2131 IOPorts are exposed.

2132

2133 **Usage:**

2134 `IOPort.getName (void): string`

2135

2136 **Parameter(s):**

2137 Data Type: `void` . This command takes no parameters.

2138

2139 **Return Value(s):**

2140 Data Type: `string` . Reader will return the name of this IO-port.

2141

2142 **Possible Error Conditions:**

2143 `ERROR_IOPORT_NOT_FOUND`

2144 `ERROR_COMMAND_NOT_SUPPORTED`

2145

### 2146 **5.8.2 IOPort.getDescription**

2147 This operation provides a textual description of the IO-port. This is typically used to  
2148 denote the equipment connected via this IO-port.

2149 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2150 IOPorts are exposed.

2151

2152 **Usage:**  
2153 `IOPort.getDescription (void): string`

2154

2155 **Parameter(s):**

2156 Data Type: `void`. This command takes no parameters.

2157

2158 **Return Value(s):**

2159 Data Type: `string`. Reader will return the description of the IO-port in a free form  
2160 String. If no description has been set the operation returns a String of length zero (an  
2161 empty string).

2162

2163 **Possible Error Conditions:**

2164 `ERROR_IOPORT_NOT_FOUND`

2165 `ERROR_COMMAND_NOT_SUPPORTED`

2166

### 2167 **5.8.3 IOPort.setDescription**

2168 This operation is used to associate a textual description with an IO-port. This is typically  
2169 used to denote the equipment connected via this IO-port.

2170 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2171 IOPorts are exposed.

2172

2173 **Usage:**

2174 `IOPort.setDescription (description : string): void`

2175

2176 **Parameter(s):**

2177 `description` – Data Type: `string`. The textual description.

2178

2179 **Return Value(s):**

2180 Data Type: `void`. This operation does not provide a return value.

2181

2182 **Possible Error Conditions:**

2183 `ERROR_IOPORT_NOT_FOUND`

2184 `ERROR_PARAMETER_LENGTH_EXCEEDED`

2185 `ERROR_COMMAND_NOT_SUPPORTED`

2186

#### 2187 **5.8.4 IOPort.getOperStatus**

2188 The Host queries the Reader for the operational status of this particular IO-port. The  
2189 operational status reported is one of the values in the OperationalStatus enumeration, e.g.  
2190 UP, DOWN, UNKNOWN or OTHER. The operational status represents the actual status  
2191 of the IOPort. It may be polled using getOperStatus or monitored via alarms (see  
2192 getOperStatusAlarmControl).

2193

2194 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2195 IOPorts are exposed.

2196

#### 2197 **Usage:**

2198 `IOPort.getOperStatus (void): OperationalStatus`

2199

#### 2200 **Parameter(s):**

2201 Data Type: `void`. This command takes no parameters.

2202

#### 2203 **Return Value(s):**

2204 Data Type: `OperationalStatus`. Reader will return the current operational status of  
2205 this IO-port.

2206

#### 2207 **Possible Error Conditions:**

2208 `ERROR_IOPORT_NOT_FOUND`

2209 `ERROR_COMMAND_NOT_SUPPORTED`

2210

#### 2211 **5.8.5 IOPort.getAdminStatus**

2212 The Host queries the Reader for the administrative status of a particular IO-port. The  
2213 administrative status reported is one of the values in the AdministrativeStatus  
2214 enumeration, e.g. UP or DOWN.

2215 The administrative status represents the host's *desired* status for this IOPort. This differs  
2216 from the operational status which represents the *actual* IOPort status. The administrative  
2217 status is set via a call to `setAdminStaus` and `getAdminStatus` simply returns the most-  
2218 recently-set value.

2219

2220 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2221 IOPorts.

2222

2223 **Usage:**

2224 `IOPort.getAdminStatus (void): AdministrativeStatus`

2225

2226 **Parameter(s):**

2227 Data Type: `void`. This command takes no parameters.

2228

2229 **Return Value(s):**

2230 Data Type: `AdministrativeStatus`. Reader will return the current administrative  
2231 status of this IO-port.

2232

2233 **Possible Error Conditions:**

2234 `ERROR_IOPORT_NOT_FOUND`

2235 `ERROR_COMMAND_NOT_SUPPORTED`

2236

## 2237 **5.8.6 IOPort.setAdminStatus**

2238 This operation is used to set the administrative status of a particular IO-port. The  
2239 administrative status assigned is one of the values in the `AdministrativeStatus`  
2240 enumeration, e.g. `UP` or `DOWN`

2241 The administrative status represents the host's *desired* status for this IOPort. This differs  
2242 from the operational status which represents the *actual* IOPort status. Hosts may query  
2243 the most-recently-requested status using `getAdminStatus`.

2244

2245 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2246 IOPorts are exposed.

2247

2248 **Usage:**

2249 `IOPort.setAdminStatus (administrativeStatus :  
2250 AdministrativeStatus): void`

2251

2252 **Parameter(s):**

2253 `administrativeStatus` – Data Type: `AdministrativeStatus`.. This is the  
2254 new administrative status to be set.

2255

2256 **Return Value(s):**

2257 Data Type: void. This operation does not provide a return value.

2258

2259 **Possible Error Conditions:**

2260 ERROR\_COMMAND\_NOT\_SUPPORTED

2261 ERROR\_IOPORT\_NOT\_FOUND

2262 ERROR\_INVALID\_ADMIN\_STATUS

2263

### 2264 **5.8.7 IOPort.getOperStatusAlarmControl**

2265 The Host queries the IOPort object for its OperStatusAlarmControl attribute of  
2266 type TTOperationalStatusAlarmControl. This attribute is the object that  
2267 controls the conditions for generating alarms alerting a manager of changes in an  
2268 IOPort's operational status.

2269 In addition to an alarm, the operational status may also be polled (see getOperStatus).

2270

2271 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2272 AlarmChannel is implemented..

2273

2274 **Usage:**

2275 IOPort.getOperStatusAlarmControl (void):

2276 TTOperationalStatusAlarmControl

2277

2278 **Parameter(s):**

2279 Data Type: void. This command takes no parameters.

2280

2281 **Return Value(s):**

2282 Data Type: TTOperationalStatusAlarmControl. An alarm control for  
2283 monitoring the operational status of the IOPort.

2284

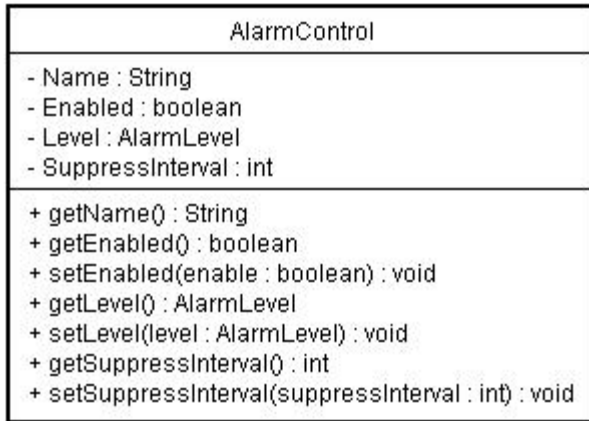
2285 **Possible Error Conditions:**

2286 ERROR\_COMMAND\_NOT\_SUPPORTED

2287 ERROR\_IOPORT\_NOT\_FOUND

2288

2289 AlarmControl ObjectAlarmControl is the base class for all classes responsible for  
2290 controlling the generation of Alarm messages, and consists of four main attributes.



2291

2292 **Figure 15 AlarmControl UML**

2293

2294

2295



2296 The Name attribute (of type String) is a unique identifier of an AlarmControl  
 2297 object.

2298 The Enabled attribute (of type Boolean) controls whether the Alarm is enabled. If  
 2299 set to false, Alarm generation will be inhibited for the specific Alarm this object  
 2300 controls.

2301 The Level attribute (of type AlarmLevel) specifies the Alarm level (defined by the  
 2302 AlarmLevel enumeration) assigned to alarms whose generation this object controls.

2303 The SuppressInterval attribute (of type integer) specifies the minimum number  
 2304 of seconds that SHALL elapse before the next Alarm of the same type is generated.  
 2305 Each object can only generate one of these Alarms per SupressInterval. Setting  
 2306 the SuppressInterval to 0 results in an Alarm generated for every Alarm condition  
 2307 encountered. Setting SuppressInterval to a value greater than zero reduces the  
 2308 number of Alarms raised, minimizing the network traffic and load on the health  
 2309 monitoring applications. The ideal value will depend on the monitored environment and  
 2310 how frequently the health monitoring application needs to be interrupted to process  
 2311 Alarm conditions. Every Alarm contains a SuppressCount indicating the number  
 2312 of times the Alarm generation has been suppressed during SuppressInterval. This count  
 2313 is reset to 0 after the successful generation of the Alarm.

2314 For example, consider two AntennaReadPoint instances named AntennaReadPoint1  
 2315 and AntennaReadPoint2. The FailedKillAlarmControl and  
 2316 FailedWriteAlarmControl have both been enabled with SupressInterval  
 2317 set to 15 seconds. The following table lists the action taken when a number of Alarm  
 2318 conditions are encountered at different times.

2319



Time	Object	Alarm Condition	Action
09:00:00	AntennaReadPoint1	Kill Failure	Generate Alarm
09:00:01	AntennaReadPoint2	Kill Failure	Generate Alarm
09:00:05	AntennaReadPoint1	Kill Failure	Suppress Alarm
09:00:06	AntennaReadPoint1	Write Failure	Generate Alarm
09:00:08	AntennaReadPoint1	Kill Failure	Suppress Alarm
09:00:16	AntennaReadPoint1	Kill Failure	Generate Alarm

2320

2321

2322 Defined subclasses of AlarmControl are:

2323 • EdgeTriggeredAlarmControl

2324 •

2325 • TTOperationalStatusAlarmControl

2326 Future revisions of this specification may define new AlarmControls in addition to

2327 EdgeTriggeredAlarmControl and TTOperationStatusAlarmControl

2328

### 2329 **5.8.8 AlarmControl.getName**

2330 Queries the reader for the current value of the AlarmControl's Name attribute.

2331

2332 **Compliance Requirement:** Compliant systems SHALL implement this command if

2333 AlarmChannel is implemented.

2334

2335 **Usage:**

2336 AlarmControl.getName (void): string

2337

2338 **Parameter(s):**

2339 Data Type: void. This command takes no parameters.

2340

2341 **Return Value(s):**

2342 Data Type: string. The AlarmControl's unique name.

2343

2344 **Possible Error Conditions:**

2345 None.

2346

### 2347 **5.8.9 AlarmControl.setEnabled**

2348 Queries the reader for the current value of the AlarmControl's Enabled attribute.

2349

2350 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2351 AlarmChannel is implemented.

2352

2353 **Usage:**

2354 AlarmControl.setEnabled(void): boolean

2355

2356 **Parameter(s):**

2357 Data Type: void. This command takes no parameters.

2358

2359 **Return Value(s):**

2360 Data Type: boolean. Indicates whether the AlarmControl is enabled.

2361

2362 **Possible Error Conditions:**

2363 None.

2364

### 2365 **5.8.10 AlarmControl.setEnabled**

2366 Enables or disables alarm generation.

2367

2368 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2369 AlarmChannel is implemented.

2370

2371 **Usage:**

2372 AlarmControl.setEnabled(enable: boolean): void

2373

2374 **Parameter(s):**

2375 enable – Data Type: boolean. A boolean that enables or disables  
2376 the AlarmControl.

2377

2378 **Return Value(s):**

2379 Data Type: void.

2380

2381 **Possible Error Conditions:**

2382 ERROR\_AUTHORIZATION. A management application may not be authorized to alter  
2383 alarm controls.

2384

### 2385 **5.8.11 AlarmControl.getLevel**

2386 Queries the reader for the current value of the AlarmControl's Level attribute.

2387 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2388 AlarmChannel is implemented.

2389

2390 **Usage:**

2391 AlarmControl.getLevel (void): AlarmLevel

2392

2393 **Parameter(s):**

2394 Data Type: void. This command takes no parameters.

2395

2396 **Return Value(s):**

2397 Data Type: AlarmLevel.

2398

2399 **Possible Error Conditions:**

2400 None.

2401

### 2402 **5.8.12 AlarmControl.setLevel**

2403 Sets the Level attribute.

2404 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2405 AlarmChannel is implemented.

2406

2407 **Usage:**

2408 AlarmControl.setLevel (alarmLevel: AlarmLevel): void

2409

2410 **Parameter(s):**

2411 alarmLevel – Data Type: AlarmLevel. The desired Alarm Level value.

2412

2413 **Return Value(s):**

2414 Data Type: void.

2415

2416 **Possible Error Conditions:**

2417 ERROR\_AUTHORIZATION. A management application may not be authorized to alter  
2418 alarm controls.

2419

### 2420 **5.8.13 AlarmControl.getSuppressInterval**

2421 Queries the reader for the current value of the AlarmControl's SuppressInterval  
2422 attribute.

2423 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2424 AlarmChannel is implemented.

2425

2426 **Usage:**

2427 AlarmControl.getSuppressInterval (void): integer

2428

2429 **Parameter(s):**

2430 Data Type: void. This command takes no parameters.

2431

2432 **Return Value(s):**

2433 Data Type: integer. The value of the AlarmControl  
2434 SuppressInterval value in msec.

2435

2436 **Possible Error Conditions:**

2437 None.

2438

### 2439 **5.8.14 AlarmControl.setSuppressInterval**

2440 Sets the SuppressInterval attribute.

2441 **Compliance Requirement:** Compliant systems SHALL implement this command if  
2442 AlarmChannel is implemented.

2443

2444 **Usage:**  
2445 AlarmControl.setSuppressInterval (suppressInterval :  
2446 integer): void

2447

2448 **Parameter(s):**

2449 suppressInterval – Data Type: integer. The desired Alarm  
2450 SuppressInterval value in msec.

2451

2452 **Return Value(s):**

2453 Data Type: void: This command will not return a value.

2454

2455 **Possible Error Conditions:**

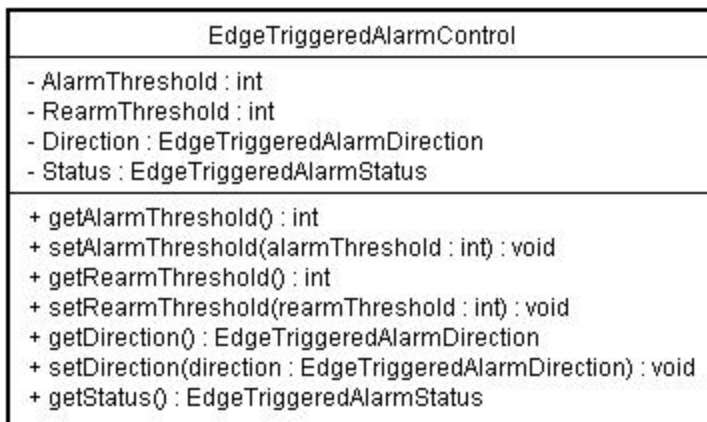
2456 ERROR\_AUTHORIZATION. A management application may not be authorized to alter  
2457 alarm controls.

2458

2459

## 2460 **5.9 EdgeTriggeredAlarmControl**

2461 This class extends AlarmControl to control alarms generated when a monitored,  
2462 integer-valued, model element first crosses a threshold value (the AlarmThreshold).  
2463 This type of alarm is called “edge triggered”. The monitored value must cross a  
2464 potentially different threshold (the RearmThreshold) before it can be triggered again.



2465

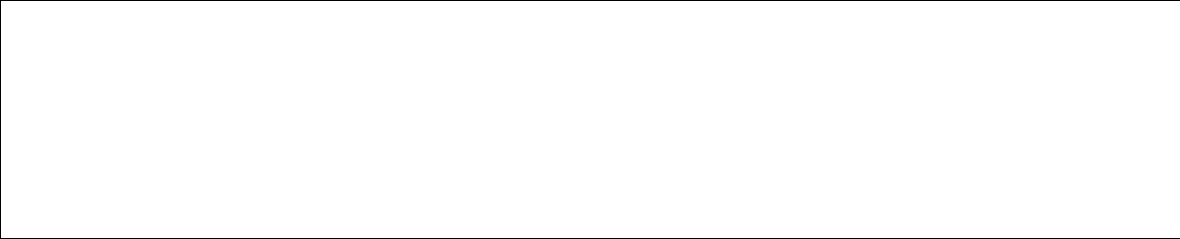
2466 **Figure 16 EdgeTriggeredAlarmControl UML**

2467

2468

2469

2470  
2471  
2472  
2473  
2474  
2475



2476 The `Direction` attribute (of enumerated type `EdgeTriggeredAlarmDirection`)  
2477 indicates whether the alarm will be triggered on an  
2478 `EdgeTriggeredAlarmDirection.RISING` or  
2479 `EdgeTriggeredAlarmDirection.FALLING` passage through the `AlarmThreshold`.  
2480 The `Status` attribute (of enumerated type `EdgeTriggeredAlarmStatus`) identifies  
2481 the current state of the alarm, i.e., `EdgeTriggeredAlarmStatus.ARMED` or  
2482 `EdgeTriggeredAlarmStatus.FIRED`.

2483

2484 If alarm control's direction is `RISING`, an alarm is triggered when the monitored value  
2485 first becomes greater than the `AlarmThreshold`; once triggered, the alarm enters the  
2486 `SUSPENDED` state, remaining in that state until the monitored value moves below (less  
2487 than) the `RearmThreshold`, at which time the alarm re-enters the `ARMED` state.

2488

2489 If the alarm control's direction is `FALLING`, an alarm is triggered when the monitored  
2490 value first becomes less than the `AlarmThreshold`; once triggered, the alarm enters the  
2491 `SUSPENDED` state, remaining in that state until the monitored value moves above  
2492 (greater than) the `RearmThreshold`, at which time the alarm re-enters the `ARMED`  
2493 state.

### 2494 **5.9.1 EdgeTriggeredAlarmControl.getAlarmThreshold**

2495 Queries the reader for the current value of the `EdgeTriggeredAlarmControl`'s  
2496 `AlarmThreshold` attribute.

2497

2498 **Compliance Requirement:** Compliant systems SHALL implement this command.

2499

2500 **Usage:**

2501 `EdgeTriggeredAlarmControl.getAlarmThreshold(void): integer`

2502

2503 **Parameter(s):**

2504 Data Type: `void`. This command takes no parameters.

2505

2506 **Return Value(s):**

2507 Data Type: integer .

2508

2509 **Possible Error Conditions:**

2510 None.

## 2511 **5.9.2 EdgeTriggeredAlarmControl.setAlarmThreshold**

2512 Command sent to the reader to set the current value of the  
2513 EdgeTriggeredAlarmControl's AlarmThreshold attribute.

2514

2515 **Compliance Requirement:** Compliant systems SHALL implement this command.

2516

2517 **Usage:**

2518 EdgeTriggeredAlarmControl.setAlarmThreshold (alarmThreshold:  
2519 integer): void

2520

2521 **Parameter(s):**

2522 alarmThreshold – Data Type: integer . The desired value of AlarmThreshold.

2523

2524 **Return Value(s):**

2525 Data Type: void.

2526

2527 **Possible Error Conditions:**

2528 ERROR\_AUTHORIZATION. A management application may not be authorized to alter  
2529 alarm controls.

2530

### 2531 **5.9.3 EdgeTriggeredAlarmControl.getRearmThreshold**

2532 Queries the reader for the current value of the EdgeTriggeredAlarmControl's  
2533 RearmThreshold attribute.

2534

2535 **Compliance Requirement:** Compliant systems SHALL implement this command.

2536

2537 **Usage:**

2538 `EdgeTriggeredAlarmControl.getRearmThreshold (void): integer`

2539

2540 **Parameter(s):**

2541 Data Type: void. This command takes no parameters.

2542

2543 **Return Value(s):**

2544 Data Type: integer.

2545

2546 **Possible Error Conditions:**

2547 None.

### 2548 **5.9.4 EdgeTriggeredAlarmControl.setRearmThreshold**

2549 Command sent to the reader to set the current value of the  
2550 EdgeTriggeredAlarmControl's RearmThreshold attribute.

2551

2552 **Compliance Requirement:** Compliant systems SHALL implement this command.

2553

2554 **Usage:**

2555 `EdgeTriggeredAlarmControl.setRearmThreshold (alarmThreshold:  
2556 integer): void`

2557

2558 **Parameter(s):**

2559 alarmThreshold – Data Type: integer. The desired value of RearmThreshold.

2560

2561 **Return Value(s):**

2562 Data Type: void.



2563

2564 **Possible Error Conditions:**

2565 ERROR\_AUTHORIZATION. A management application may not be authorized to alter  
2566 alarm controls.

2567

### 2568 **5.9.5 EdgeTriggeredAlarmControl.getDirection**

2569 Queries the reader for the current value of the EdgeTriggeredAlarmControl's  
2570 Direction attribute.

2571

2572 **Compliance Requirement:** Compliant systems SHALL implement this command.

2573

2574 **Usage:**

2575 EdgeTriggeredAlarmControl.getDirection (void):  
2576 EdgeTriggeredAlarmDirection

2577

2578 **Parameter(s):**

2579 Data Type: void. This command takes no parameters.

2580

2581 **Return Value(s):**

2582 Data Type: EdgeTriggeredAlarmDirection. The return data type is an  
2583 enumerated type with possible values RISING and FALLING.

2584

2585 **Possible Error Conditions:**

2586 None.

### 2587 **5.9.6 EdgeTriggeredAlarmControl.setDirection**

2588 Command sent to the reader to set the current value of the  
2589 EdgeTriggeredAlarmControl's Direction attribute.

2590

2591 **Compliance Requirement:** Compliant systems SHALL implement this command.

2592

2593 **Usage:**

2594 EdgeTriggeredAlarmControl.setDirection(direction:  
2595 EdgeTriggeredAlarmDirection): void

2596

2597 **Parameter(s):**  
2598 direction: – Data Type: EdgeTriggeredAlarmDirection. The desired value of  
2599 Direction.  
2600  
2601 **Return Value(s):**  
2602 Data Type: void.  
2603  
2604 **Possible Error Conditions:**  
2605 ERROR\_AUTHORIZATION. A management application may not be authorized to alter  
2606 alarm controls.  
2607  
2608

### 2609 **5.9.7 EdgeTriggeredAlarmControl.getStatus**

2610 Queries the reader for the current value of the EdgeTriggeredAlarmControl's  
2611 Status attribute.

2612

2613 **Compliance Requirement:** Compliant systems SHALL implement this command.

2614

#### 2615 **Usage:**

2616 EdgeTriggeredAlarmControl.getStatus (void):  
2617 EdgeTriggeredAlarmStatus

2618

#### 2619 **Parameter(s):**

2620 Data Type: void. This command takes no parameters.

2621

#### 2622 **Return Value(s):**

2623 Data Type: EdgeTriggeredAlarmStatus. The return data type is  
2624 an enumerated type with possible values ARMED OR FIRED.

2625

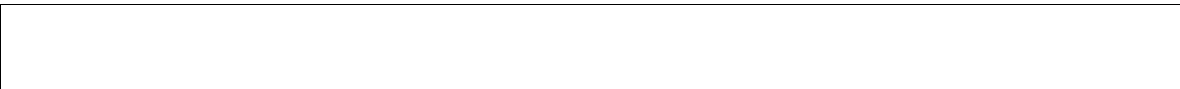
#### 2626 **Possible Error Conditions:**

2627 None.

2628

2629

2630



2631  
2632  
2633  
2634  
2635  
2636  
2637  
2638  
2639  
2640  
2641  
2642  
2643  
2644  
2645  
2646  
2647  
2648  
2649  
2650



2651 **5.10 TTOperationalStatusAlarmControl**

2652 This class extends `AlarmControl` to control alarms generated when a monitored model  
2653 element of type `OperationalStatus` transitions to a new value. This type of alarm is  
2654 called “transition triggered” (abbreviated “TT”).

2655  
2656  
2657  
2658



TTOperationalStatusAlarmControl
- TriggerFromState : OperationalStatus - TriggerToState : OperationalStatus
+ getTriggerFromState() : OperationalStatus + setTriggerFromState(triggerFromState : OperationalStatus) : void + getTriggerToState() : OperationalStatus + setTriggerToState(triggerToState : OperationalStatus) : void

2659

2660 **Figure 17** TTOperationalStatusAlarmControl UML

2661

2662 The TriggerFromState attribute (of enumerated type OperationalStatus)  
2663 indicates the value of the monitored, OperationalStatus-valued, model element prior  
2664 to the state transition which triggers the alarm. The TriggerToState attribute (of  
2665 enumerated type OperationalStatus) indicates the value of the monitored,  
2666 OperationalStatus-valued, model element immediately after the state transition  
2667 which triggers the alarm.

2668

### 2669 **5.10.1** **TTOperationalStatusAlarmControl.getTriggerFromSta** 2670 **te**

2671 Queries the reader for the current value of the  
2672 TTOperationalStatusAlarmControl's TriggerFromState attribute.

2673

2674 **Compliance Requirement:** Compliant systems SHALL implement this command.

2675

2676 **Usage:**

2677 `TTOperationalStatusAlarmControl.getTriggerFromState (void) :`  
2678 `OperationalStatus`

2679

2680 **Parameter(s):**

2681 Data Type: void. This command takes no parameters.

2682

2683 **Return Value(s):**

2684 Data Type: OperationalStatus. The return data type is an enumerated type with  
2685 possible values UNKNOWN, OTHER, UP, DOWN and ANY.

2686 **Possible Error Conditions:**

2687 None.

2688 **5.10.2 TTOperationalStatusAlarmControl.setTriggerFromSta**  
2689 **te**

2690 Command sent to the reader to set the current value of the  
2691 TTOperationalStatusAlarmControl's TriggerFromState attribute.

2692

2693 **Compliance Requirement:** Compliant systems SHALL implement this command.

2694

2695 **Usage:**

2696 TTOperationalStatusAlarmControl.setTriggerFromState  
2697 (triggerFromState: OperationalStatus): void

2698

2699 **Parameter(s):**

2700 operationalStatus – Data Type: OperationalStatus. The desired value of  
2701 TriggerFromState.

2702

2703 **Return Value(s):**

2704 Data Type: void.

2705

2706 **Possible Error Conditions:**

2707 ERROR\_AUTHORIZATION. A management application may not be authorized to alter  
2708 alarm controls.

2709

2710 **5.10.3 TTOperationalStatusAlarmControl.getTriggerToState**

2711 Queries the reader for the current value of the  
2712 TTOperationalStatusAlarmControl's TriggerToState attribute.

2713

2714 **Compliance Requirement:** Compliant systems SHALL implement this command.

2715

2716 **Usage:**

2717 TTOperationalStatusAlarmControl.getTriggerToState (void):  
2718 OperationalStatus

2719

2720 **Parameter(s):**

2721 Data Type: void. This command takes no parameters.

2722

2723 **Return Value(s):**

2724 Data Type: `OperationalStatus`. The return data type is an enumerated type with  
2725 possible values UNKNOWN, OTHER, UP, DOWN and ANY.

2726 **Possible Error Conditions:**

2727 None.

2728 **5.10.4 TOperationalStatusAlarmControl.setTriggerToState**

2729 Command sent to the reader to set the current value of the  
2730 TOperationalStatusAlarmControl's TriggerToState attribute.

2731

2732 **Compliance Requirement:** Compliant systems SHALL implement this command.

2733

2734 **Usage:**

2735 `TOperationalStatusAlarmControl.setTriggerToState`  
2736 `(triggerToState: OperationalStatus): void`

2737

2738 **Parameter(s):**

2739 `operationalStatus` – Data Type: `OperationalStatus`. The desired value of  
2740 `TriggerToState`.

2741

2742 **Return Value(s):**

2743 Data Type: `void`.

2744

2745 **Possible Error Conditions:**

2746 `ERROR_AUTHORIZATION`. A management application may not be authorized to alter  
2747 alarm controls.

2748

2749

2750

2751

2752

2753

2754

2755



2756  
2757  
2758  
2759  
2760  
2761  
2762  
2763  
2764  
2765  
2766  
2767  
2768  
2769  
2770  
2771  
2772  
2773  
2774  
2775  
2776  
2777  
2778

## 2779 **6 Reader Layer – Alarm Notifications**

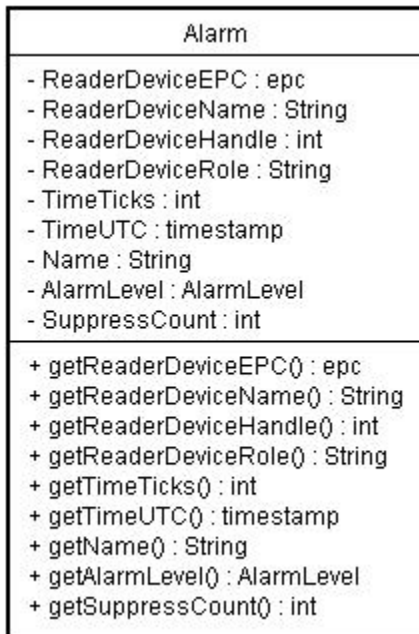
### 2780 **6.1 Alarm Objects**

2781 The AlarmControl class, and its subclasses, specify those elements of a Reader’s object  
2782 model that control alarm generation. The alarms themselves are messages that are sent to  
2783 management systems and host applications whose responsibility it is to monitor Reader  
2784 health and operations. The Reader object model has been expanded to include class  
2785 definitions that specify, in an abstract manner, the contents of these alarm messages.  
2786 Note that these Alarm objects do not specify persistent data objects maintained by the  
2787 readers. These objects are instantiated in the course of a reader’s generation or a  
2788 management system’s receipt and processing of alarms; once an alarm message is  
2789 transmitted or processed, its corresponding information object disappears.

2790 In the object descriptions below, the UML for each object is presented with its elements  
2791 and operations or methods. Each object element has an associated getter. Since Alarms  
2792 can only be generated by the reader device and can not be modified by the consumer, no  
2793 setters are provided. Getters are used during message processing.  
2794

## 2795 6.2 Alarm

2796 Alarm is the base of all the classes within the object model that define the contents of  
2797 alarm messages.



2798

2799 **Figure 18 Alarm UML**

2800

2801 Each Alarm MAY carry a ReaderDeviceEPC attribute (of type epc)  
2802 specifying the EPC of the reader that generated the alarm. The  
2803 reader EPC is ReaderDevice.EPC element of the Object model.

2804

2805

2806

2807

2808

2809



2810 Each Alarm SHALL carry a `ReaderDeviceName` attribute (of type `String`)  
 2811 specifying the name of the reader that generated the alarm. The reader name is  
 2812 `ReaderDevice.Name` element of the Object model.

2813 Each Alarm MAY carry a `ReaderDeviceHandle` attribute (of type `integer`)  
 2814 specifying the handle of the reader that generated the alarm. The reader handle is  
 2815 `ReaderDevice.Handle` element of the Object model.

2816 Each Alarm MAY carry a `ReaderDeviceRole` attribute (of type `String`)  
 2817 specifying the Role of the reader that generated the alarm. The reader Role is  
 2818 `ReaderDevice.Role` element of the object model.

2819 Each Alarm SHALL also carry a `TimeTicks` attribute (of type `integer`) recording the  
 2820 number of ticks when the alarm was generated. It's value is the value of the  
 2821 `ReaderDevice.TimeTicks` element of the Object model at the time the Alarm was  
 2822 generated.

2823 Each Alarm MAY also carry a `TimeUTC` attribute (of type `timestamp`) recording when  
 2824 the alarm was generated. It's value is the value of the `ReaderDevice.TimeUTC`  
 2825 element of the object model at the time the Alarm was generated.

2826 Each Alarm SHALL carry a `Name` attribute (of type `String`) identifying the type of  
 2827 Alarm, e.g., “FreeMemoryAlarm”, “TagListFullAlarm”, “ReadPointOperStatusAlarm”.  
 2828 It SHALL matched on of the Alarm types specified in this section in this  
 2829 specification. Each Alarm SHALL carry an `AlarmLevel` attribute (of type  
 2830 `AlarmLevel`), indicating the severity level assigned to the alarm.

2831 Each Alarm SHALL carry a `SuppressCount` attribute (of type `integer`), indicating the  
 2832 number of times the generation of this Alarm has been suppressed. This attribute is  
 2833 incremented every time an alarm condition is encountered while the  
 2834 `SuppressInterval` is in effect, preventing the reader from generating the Alarm.  
 2835 This value is reset to 0 after the Alarm is generated. Suppression is controlled by the  
 2836 `SuppressInterval` attribute of `AlarmControl`.

2837 Expanding on the example presented in the `AlarmControl` section, a  
 2838 `FailedLockAlarm` is generated at 09:00:00. At 09:00:05, the alarm condition is  
 2839 encountered again while the `SuppressInterval` has not yet been reached. At this  
 2840 time, the `SuppressCount` for `AntennaReadPoint1.FailedKillAlarmControl` is  
 2841 incremented by one and the alarm generation is suppressed. The same occurs at  
 2842 09:00:08, incrementing `SuppressCount` by 1 again while suppressing the Alarm. At  
 2843 09:00:16, the Alarm is generated containing a `SuppressCount` equal to 2, indicating  
 2844 the Alarm was not sent on 2 prior occasions during the interval from 09:00:00 (first  
 2845 alarm) until now. The `SuppressCount` for this object will then be reset to 0 after the  
 2846 successful generation of the Alarm.

2847

Time	Object	Alarm Condition	Action	Suppress Count
------	--------	-----------------	--------	----------------

09:00:00	AntennaReadPoint1	Kill Failure	Generate Alarm	0
09:00:01	AntennaReadPoint2	Kill Failure	Generate Alarm	0
09:00:05	AntennaReadPoint1	Kill Failure	Suppress Alarm	Increment
09:00:06	AntennaReadPoint1	Write Failure	Generate Alarm	0
09:00:08	AntennaReadPoint1	Kill Failure	Suppress Alarm	Increment
09:00:16	AntennaReadPoint1	Kill Failure	Generate Alarm	2

2848

### 2849 **6.2.1 Alarm.getReaderDeviceEPC**

2850 Queries the Alarm for the value of the ReaderDeviceEPC attribute.

2851

2852 **Compliance Requirement:** If this alarm is implemented, compliant systems MAY  
2853 implement this command.

2854

2855 **Usage:**

2856 Alarm.getReaderDeviceEPC (void): epc

2857

2858 **Parameter(s):**

2859 Data Type: void. This command takes no parameters.

2860

2861 **Return Value(s):**

2862 Data Type: epc. The EPC for the reader device.

2863

2864 **Possible Error Conditions:**

2865 None.

### 2866 **6.2.2 Alarm.getReaderDeviceName**

2867 Queries the Alarm for the value of the ReaderDeviceName attribute.

2868

2869 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
2870 implement this command.

2871

2872 **Usage:**

2873 Alarm.getReaderDeviceName (void): string

2874

2875 **Parameter(s):**

2876 Data Type: `void`. This command takes no parameters.

2877

2878 **Return Value(s):**

2879 Data Type: `string`. The name for the reader device.

2880

2881 **Possible Error Conditions:**

2882 None.

### 2883 **6.2.3 Alarm.getReaderDeviceHandle**

2884 Queries the Alarm for the value of the `ReaderDeviceHandle` attribute.

2885

2886 **Compliance Requirement:** Compliant systems MAY implement this command.

2887

2888 **Usage:**

2889 `Alarm.getReaderDeviceHandle (void): integer`

2890

2891 **Parameter(s):**

2892 Data Type: `void`. This command takes no parameters.

2893

2894 **Return Value(s):**

2895 Data Type: `integer`. The handle for the reader device.

2896

2897 **Possible Error Conditions:**

2898 None.

### 2899 **6.2.4 Alarm.getReaderDeviceRole**

2900 Queries the Alarm for the value of the `ReaderDeviceRole` attribute.

2901

2902 **Compliance Requirement:** Compliant systems MAY implement this command.

2903

2904 **Usage:**

2905 `Alarm.getReaderDeviceRole (void): string`

2906

2907 **Parameter(s):**

2908 Data Type: void. This command takes no parameters.

2909

2910 **Return Value(s):**

2911 Data Type: string. The role for the reader device.

2912

2913 **Possible Error Conditions:**

2914 None.

### 2915 **6.2.5 Alarm.getTimeTicks**

2916 Queries the Alarm for the value of the TimeTicks attribute.

2917

2918 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
2919 implement this command.

2920

2921 **Usage:**

2922 Alarm.getTimeTicks (void): integer

2923

2924 **Parameter(s):**

2925 Data Type: void. This command takes no parameters.

2926

2927 **Return Value(s):**

2928 Data Type: integer. The TimeTicks for the reader device.

2929

2930 **Possible Error Conditions:**

2931 None.

### 2932 **6.2.6 Alarm.getTimeUTC**

2933 Queries the Alarm for the value of the TimeUTC attribute.

2934

2935 **Compliance Requirement:** Compliant systems MAY implement this command.

2936

2937 **Usage:**

2938 Alarm.getTimeUTC (void): timestamp

2939

2940 **Parameter(s):**

2941 Data Type: void. This command takes no parameters.

2942

2943 **Return Value(s):**

2944 Data Type: timestamp. The UTC time for the reader device.

2945

2946 **Possible Error Conditions:**

2947 None.

2948 **6.2.7 Alarm.getName**

2949 Queries the Alarm for the value of the Name attribute.

2950

2951 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
2952 implement this command.

2953

2954 **Usage:**

2955 Alarm.getName (void): string

2956

2957 **Parameter(s):**

2958 Data Type: void. This command takes no parameters.

2959

2960 **Return Value(s):**

2961 Data Type: string. The name for the alarm.

2962

2963 **Possible Error Conditions:**

2964 None.

2965 **6.2.8 Alarm.getAlarmLevel**

2966 Queries the Alarm for the value of the AlarmLevel attribute.

2967

2968 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
2969 implement this command.

2970

2971 **Usage:**

2972 Alarm.getAlarmLevel (void): AlarmLevel

2973

2974 **Parameter(s):**

2975 Data Type: void. This command takes no parameters.

2976

2977 **Return Value(s):**

2978 Data Type: AlarmLevel. The alarm severity level.

2979

2980 **Possible Error Conditions:**

2981 None.

## 2982 **6.2.9 Alarm.getSuppressCount**

2983 Queries the Alarm for the value of the SuppressCount attribute.

2984

2985 **Compliance Requirement:** Compliant systems SHALL implement this command.

2986

2987 **Usage:**

2988 Alarm.getSuppressCount (void): integer

2989

2990 **Parameter(s):**

2991 Data Type: void. This command takes no parameters.

2992

2993 **Return Value(s):**

2994 Data Type: integer. The number of times since last alarm  
2995 generation that this alarm has been suppressed from being  
2996 generated.

2997

2998 **Possible Error Conditions:**

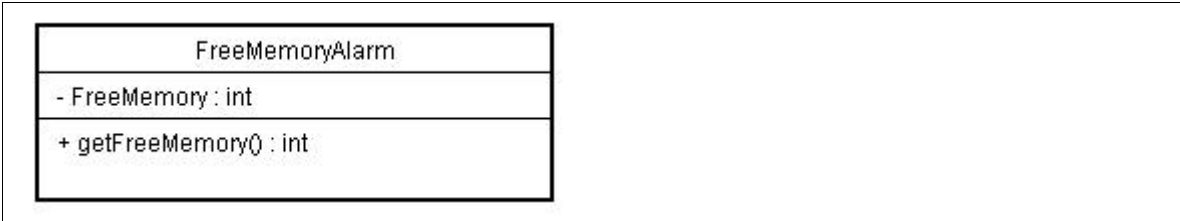
2999 None.

3000

3001  
3002  
3003  
3004  
3005  
3006  
  
3007  
3008  
  
3009  
3010  
3011  
3012  
3013  
  
3014  
3015  
3016  
3017  
3018  
3019  
3020  
3021  
3022  
3023  
3024  
3025  
3026  
3027  
3028  
3029  
3030

### 6.3 FreeMemoryAlarm

FreeMemoryAlarm extends the Alarm class. Its receipt signals the movement of a reader device's free memory (represented in the abstract model by ReaderDevice.FreeMemory) below a specified threshold value. The abstract model's ReaderDevice.FreeMemoryAlarmControl object controls the triggering of alarms of this type.



**Figure 19 FreeMemoryAlarm UML**

The FreeMemory attribute SHALL carry the value of ReaderDevice.FreeMemory when the alarm was triggered.

**Compliance Requirement:** Compliant systems MAY implement this alarm.

#### 6.3.1 FreeMemoryAlarm.getFreeMemory

Queries the FreeMemoryAlarm for the value of the FreeMemory attribute.

**Compliance Requirement:** If this alarm is implemented, compliant systems SHALL implement this command.

**Usage:**

FreeMemoryAlarm.getFreeMemory (void): integer

**Parameter(s):**

Data Type: void. This command takes no parameters.

**Return Value(s):**

Data Type: integer. The free memory for the reader device.

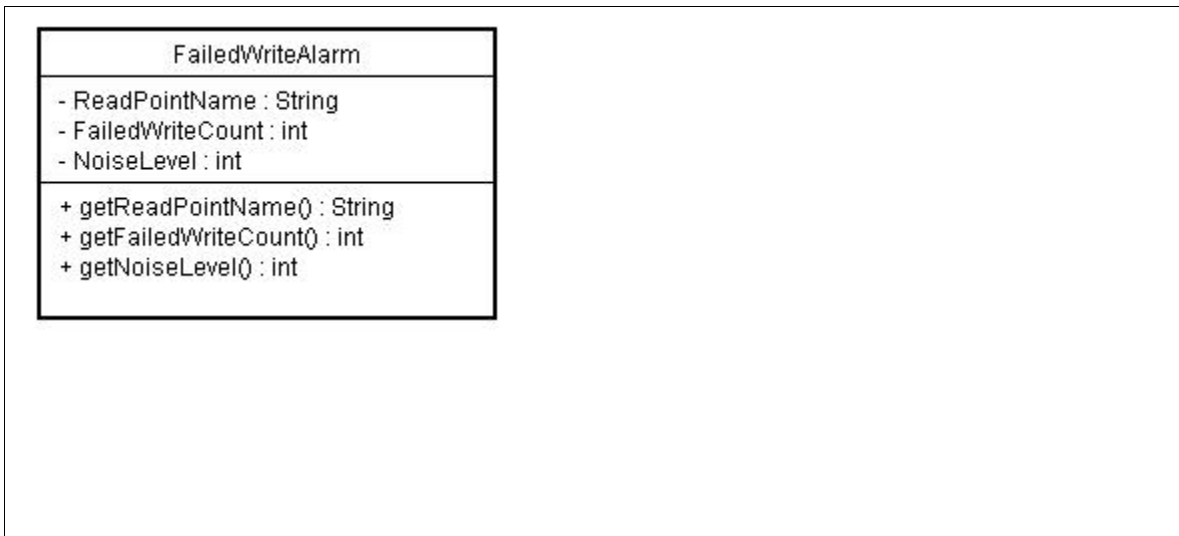
**Possible Error Conditions:**

None.

3031

## 3032 6.4 FailedWriteAlarm

3033 FailedWriteAlarm extends the Alarm class. Its receipt signals a tag write or tag block  
3034 write failure. The abstract model's  
3035 AntennaReadPoint.FailedWriteAlarmControl data element controls the  
3036 triggering of alarms of this type.



3037

3038

3039

3040

3041

3042

3043 **Figure 20 FailedWriteAlarm UML**

3044

3045 The ReadPointName attribute identifies the read point (an AntennaReadPoint) over  
3046 which the write failure occurred. This SHALL be the value of that read point's  
3047 ReadPoint.Name element.

3048 The FailedWriteCount attribute SHALL carry the value of  
3049 AntennaReadPoint.FailedWriteCount element after the write failure occurred.

3050 The NoiseLevel attribute MAY carry the value of the  
3051 AntennaReadPoint.NoiseLevel element when the write failure occurred.

3052 **Compliance Requirement:** Compliant systems MAY implement this alarm.

### 3053 6.4.1 FailedWriteAlarm.getReadPointName

3054 Queries the FailedWriteAlarm for the value of the ReadPointName attribute.

3055

3056 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3057 implement this command.

3058



3059 **Usage:**  
3060 `FailedWriteAlarm.getReadPointName (void): string`

3061

3062 **Parameter(s):**

3063 Data Type: `void`. This command takes no parameters.

3064

3065 **Return Value(s):**

3066 Data Type: `string`. The name of the `ReadPoint`.

3067

3068 **Possible Error Conditions:**

3069 None.

### 3070 **6.4.2 FailedWriteAlarm.getFailedWriteCount**

3071 Queries the `FailedWriteAlarm` for the value of the `FailedWriteCount` attribute.

3072

3073 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3074 implement this command.

3075

3076 **Usage:**

3077 `FailedWriteAlarm.getFailedWriteCount (void): integer`

3078

3079 **Parameter(s):**

3080 Data Type: `void`. This command takes no parameters.

3081

3082 **Return Value(s):**

3083 Data Type: `integer`. The `FailedWriteCount` for the `ReadPoint`.

3084

3085 **Possible Error Conditions:**

3086 None.

### 3087 **6.4.3 FailedWriteAlarm.getNoiseLevel**

3088 Queries the `FailedWriteAlarm` for the value of the `NoiseLevel` attribute.

3089

3090 **Compliance Requirement:** Compliant systems MAY implement this command.

3091

3092 **Usage:**

3093 FailedWriteAlarm.getNoiseLevel (void): integer

3094

3095 **Parameter(s):**

3096 Data Type: void. This command takes no parameters.

3097

3098 **Return Value(s):**

3099 Data Type: integer. The NoiseLevel for the ReadPoint.

3100

3101 **Possible Error Conditions:**

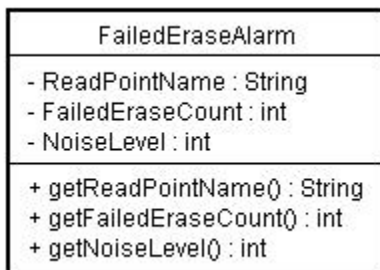
3102 None.

3103

## 3104 **6.5 FailedEraseAlarm**

3105 FailedEraseAlarm extends the Alarm class. Its receipt signals a tag erase or tag block  
3106 erase failure. The abstract model's

3107 AntennaReadPoint.FailedEraseAlarmControl data element controls the  
3108 triggering of alarms of this type.



3109

3110 **Figure 21 FailedEraseAlarm UML**

3111

3112 The ReadPointName attribute identifies the read point (an AntennaReadPoint) over  
3113 which the erase failure occurred. This SHALL be the value of that read point's  
3114 ReadPoint.Name element.

3115 The FailedEraseCount attribute SHALL carry the value of

3116 AntennaReadPoint.FailedEraseCount element after the erase failure occurred.

3117 The NoiseLevel attribute MAY carry the value of the

3118 AntennaReadPoint.NoiseLevel element when the erase failure occurred.

3119 Note that the erase failure could be attributable to the use of an incorrect tag erase  
3120 password.

3121 Compliance Requirement: Compliant systems MAY implement this alarm.

### 3122 **6.5.1 FailedEraseAlarm.getReadPointName**

3123 Queries the FailedEraseAlarm for the value of the ReadPointName attribute.

3124

3125 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3126 implement this command.

3127

3128 **Usage:**

3129 FailedEraseAlarm.getReadPointName (void): string

3130

3131 **Parameter(s):**

3132 Data Type: void. This command takes no parameters.

3133

3134 **Return Value(s):**

3135 Data Type: string. The name of the ReadPoint.

3136

3137 **Possible Error Conditions:**

3138 None.

### 3139 **6.5.2 FailedEraseAlarm.getFailedEraseCount**

3140 Queries the FailedEraseAlarm for the value of the FailedEraseCount attribute.

3141

3142 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3143 implement this command.

3144

3145 **Usage:**

3146 FailedEraseAlarm.getFailedEraseCount (void): integer

3147

3148 **Parameter(s):**

3149 Data Type: void. This command takes no parameters.

3150

3151 **Return Value(s):**

3152 Data Type: integer. The FailedEraseCount for the ReadPoint.

3153

3154 **Possible Error Conditions:**

3155 None.

### 3156 **6.5.3 FailedEraseAlarm.getNoiseLevel**

3157 Queries the FailedEraseAlarm for the value of the NoiseLevel attribute.

3158

3159 **Compliance Requirement:** Compliant systems MAY implement this command.

3160

3161 **Usage:**

3162 FailedEraseAlarm.getNoiseLevel (void): integer

3163

3164 **Parameter(s):**

3165 Data Type: void. This command takes no parameters.

3166

3167 **Return Value(s):**

3168 Data Type: integer. The NoiseLevel for the ReadPoint.

3169

3170 **Possible Error Conditions:**

3171 None.

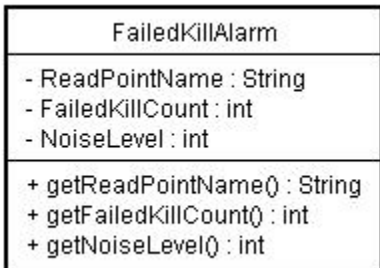
3172

## 3173 **6.6 FailedKillAlarm**

3174 FailedKillAlarm extends the Alarm class. Its receipt signals a tag kill failure. The

3175 abstract model's AntennaReadPoint.FailedKillAlarmControl data element

3176 controls the triggering of alarms of this type.



3177

3178

3179

3180

3181

3182

3183 **Figure 22 FailedKillAlarm UML**

3184

3185 The `ReadPointName` attribute identifies the read point (an `AntennaReadPoint`) over  
3186 which the kill failure occurred. This SHALL be the value of that read point's  
3187 `ReadPoint.Name` element.

3188 The `FailedKillCount` attribute SHALL carry the value of  
3189 `AntennaReadPoint.FailedKillCount` element after the kill failure occurred.

3190 The `NoiseLevel` attribute MAY carry the value of the  
3191 `AntennaReadPoint.NoiseLevel` element when the kill failure occurred.

3192 Note that the kill failure could be attributable to the use of an incorrect tag kill password.

3193 **Compliance Requirement:** Compliant systems MAY implement this alarm.

### 3194 **6.6.1 FailedKillAlarm.getReadPointName**

3195 Queries the `FailedKillAlarm` for the value of the `ReadPointName` attribute.

3196

3197 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3198 implement this command.

3199

3200 **Usage:**

3201 `FailedKillAlarm.getReadPointName (void): string`

3202

3203 **Parameter(s):**

3204 Data Type: `void`. This command takes no parameters.

3205

3206 **Return Value(s):**

3207 Data Type: `string`. The name of the `ReadPoint`.

3208

3209 **Possible Error Conditions:**

3210 None.

3211 **6.6.2 FailedKillAlarm.getFailedKillCount**

3212 Queries the FailedKillAlarm for the value of the FailedKillCount attribute.

3213

3214 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3215 implement this command.

3216

3217 **Usage:**

3218 FailedKillAlarm.getFailedKillCount (void): integer

3219

3220 **Parameter(s):**

3221 Data Type: void. This command takes no parameters.

3222

3223 **Return Value(s):**

3224 Data Type: integer. The FailedKillCount for the ReadPoint.

3225

3226 **Possible Error Conditions:**

3227 None.

3228 **6.6.3 FailedKillAlarm.getNoiseLevel**

3229 Queries the FailedKillAlarm for the value of the NoiseLevel attribute.

3230

3231 **Compliance Requirement:** Compliant systems MAY implement this command.

3232

3233 **Usage:**

3234 FailedKillAlarm.getNoiseLevel (void): integer

3235

3236 **Parameter(s):**

3237 Data Type: void. This command takes no parameters.

3238

3239 **Return Value(s):**

3240 Data Type: integer. The NoiseLevel for the ReadPoint.

3241

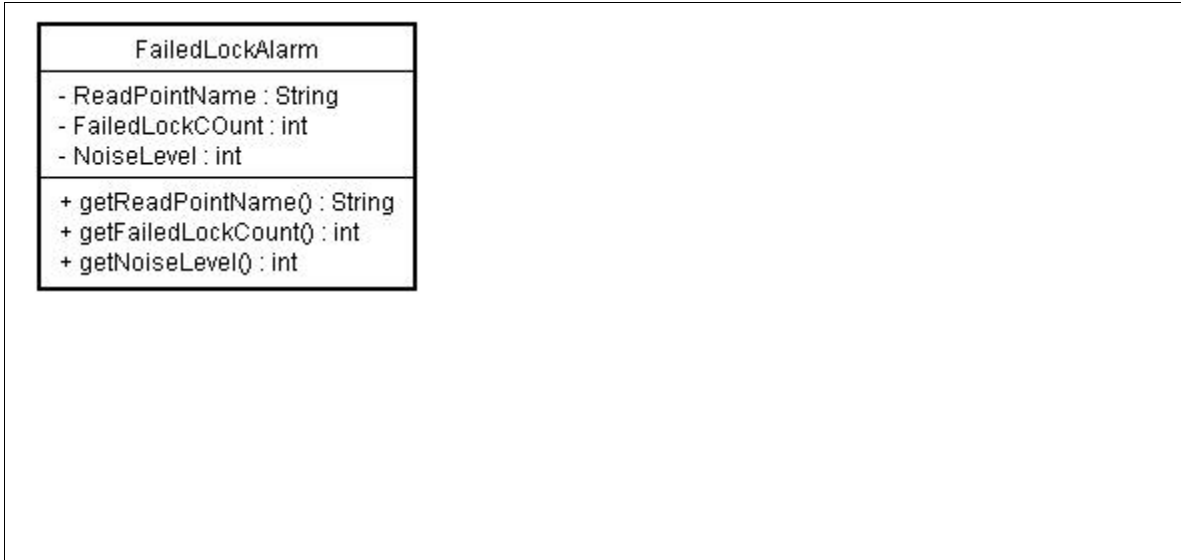
3242 **Possible Error Conditions:**

3243 None.

3244

## 3245 6.7 FailedLockAlarm

3246 FailedLockAlarm extends the Alarm class. Its receipt signals a tag lock failure. The  
3247 abstract model's AntennaReadPoint.FailedLockAlarmControl data element  
3248 controls the triggering of alarms of this type.



3249

3250

3251

3252

3253

3254

3255

3256 **Figure 23 FailedKillAlarm UML**

3257

3258 The ReadPointName attribute identifies the read point (an AntennaReadPoint) over  
3259 which the lock failure occurred. This SHALL be the value of that read point's  
3260 ReadPoint.Name element.

3261 The FailedLockCount attribute SHALL carry the value of  
3262 AntennaReadPoint.FailedLockCount element after the read failure occurred.

3263 The NoiseLevel attribute MAY carry the value of the  
3264 AntennaReadPoint.NoiseLevel element when the lock failure occurred.

3265 Note that the lock failure could be attributable to the use of an incorrect tag lock  
3266 password.

3267 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3268 implement this alarm.

### 3269 6.7.1 FailedLockAlarm.getReadPointName

3270 Queries the FailedLockAlarm for the value of the ReadPointName attribute.

3271

3272 **Compliance Requirement:** Compliant systems MAY implement this command.

3273

3274 **Usage:**

3275 `FailedLockAlarm.getReadPointName (void): string`

3276

3277 **Parameter(s):**

3278 Data Type: `void`. This command takes no parameters.

3279

3280 **Return Value(s):**

3281 Data Type: `string`. The name of the `ReadPoint`.

3282

3283 **Possible Error Conditions:**

3284 None.

### 3285 **6.7.2 FailedLockAlarm.getFailedLockCount**

3286 Queries the `FailedLockAlarm` for the value of the `FailedLockCount` attribute.

3287

3288 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3289 implement this command.

3290

3291 **Usage:**

3292 `FailedLockAlarm.getFailedLockCount (void): integer`

3293

3294 **Parameter(s):**

3295 Data Type: `void`. This command takes no parameters.

3296

3297 **Return Value(s):**

3298 Data Type: `integer`. The `FailedLockCount` for the `ReadPoint`.

3299

3300 **Possible Error Conditions:**

3301 None.

### 3302 **6.7.3 FailedLockAlarm.getNoiseLevel**

3303 Queries the `FailedLockAlarm` for the value of the `NoiseLevel` attribute.

3304



3305 **Compliance Requirement:** Compliant systems MAY implement this command.

3306

3307 **Usage:**

3308 `FailedLockAlarm.getNoiseLevel (void): integer`

3309

3310 **Parameter(s):**

3311 Data Type: `void`. This command takes no parameters.

3312

3313 **Return Value(s):**

3314 Data Type: `integer`. The `NoiseLevel` for the `ReadPoint`.

3315

3316 **Possible Error Conditions:**

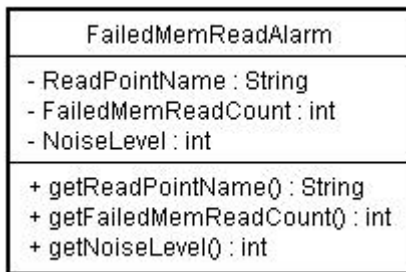
3317 None.

3318

## 3319 **6.8 FailedMemReadAlarm**

3320 `FailedMemReadAlarm` extends the `Alarm` class. Its receipt signals a tag user-memory  
3321 read failure. The abstract model's

3322 `AntennaReadPoint.FailedMemReadAlarmControl` data element controls the  
3323 triggering of alarms of this type.



3324

3325

3326

3327

3328

3329

3330

3331 **Figure 24 FailedMemReadAlarm UML**

3332

3333 The `ReadPointName` attribute identifies the read point (an `AntennaReadPoint`) over  
3334 which the memory read failure occurred. This SHALL be the value of that read point's  
3335 `ReadPoint.Name` element.

3336 The `FailedMemReadCount` attribute SHALL carry the value of  
3337 `AntennaReadPoint.FailedMemReadCount` element after the read failure occurred.

3338 The `NoiseLevel` attribute MAY carry the value of the  
3339 `AntennaReadPoint.NoiseLevel` element when the memory read failure occurred.

3340 Note that the read failure could be attributable to the use of an incorrect tag memory read  
3341 password.

3342 **Compliance Requirement:** Compliant systems MAY implement this alarm.

### 3343 **6.8.1 FailedMemReadAlarm.getReadPointName**

3344 Queries the `FailedMemReadAlarm` for the value of the `ReadPointName` attribute.

3345

3346 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3347 implement this command.

3348

3349 **Usage:**

3350 `FailedMemReadAlarm.getReadPointName (void): string`

3351

3352 **Parameter(s):**

3353 Data Type: `void`. This command takes no parameters.

3354

3355 **Return Value(s):**

3356 Data Type: `string`. The name of the `ReadPoint`.

3357

3358 **Possible Error Conditions:**

3359 None.

### 3360 **6.8.2 FailedMemReadAlarm.getFailedMemReadCount**

3361 Queries the `FailedMemReadAlarm` for the value of the `FailedMemReadCount` attribute.

3362

3363 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3364 implement this command.

3365

3366 **Usage:**

3367 FailedMemReadAlarm.getFailedMemReadCount (void): integer

3368

3369 **Parameter(s):**

3370 Data Type: void. This command takes no parameters.

3371

3372 **Return Value(s):**

3373 Data Type: integer. The FailedMemReadCount for the ReadPoint.

3374

3375 **Possible Error Conditions:**

3376 None.

3377 **6.8.3 FailedMemReadAlarm.getNoiseLevel**

3378 Queries the FailedMemReadAlarm for the value of the NoiseLevel attribute.

3379

3380 **Compliance Requirement:** Compliant systems MAY implement this command.

3381

3382 **Usage:**

3383 FailedMemReadAlarm.getNoiseLevel (void): integer

3384

3385 **Parameter(s):**

3386 Data Type: void. This command takes no parameters.

3387

3388 **Return Value(s):**

3389 Data Type: integer. The NoiseLevel for the ReadPoint.

3390

3391 **Possible Error Conditions:**

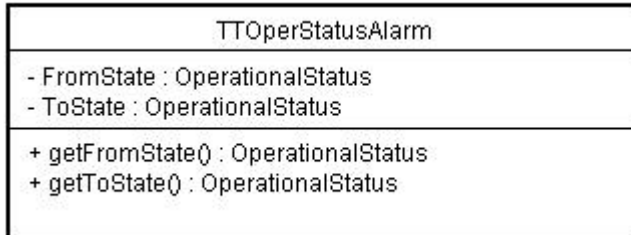
3392 None.

3393

3394

## 3395 6.9 TTOperStatusAlarm

3396 TTOperStatusAlarm extends the Alarm class, and is the base class for all Transition  
3397 Triggered Operational Status Alarms. The base class identifies the particular transition  
3398 between Operational Status states that triggered the alarm.



3399

3400 **Figure 25 TTOperStatusAlarm UML**

3401

3402 The FromState attribute SHALL identify the originating OperationalStatus before  
3403 the Alarm is generated.

3404 The ToState attribute SHALL identify the OperationalStatus at the time the  
3405 Alarm is generated.

3406 Valid values for OperationalStatus are defined in 7.1.2  
3407 OperationalStatus.

3408

3409

3410

### 3411 6.9.1 TTOperStatusAlarm.getFromState

3412 Queries the TTOperStatusAlarm for the value of the OperationalStatus attribute  
3413 before the transition.

3414

3415 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3416 implement this command.

3417

3418 **Usage:**

3419 TTOperStatusAlarm.getFromState (void): OperationalStatus

3420

3421 **Parameter(s):**

3422 Data Type: void. This command takes no parameters.

3423

3424 **Return Value(s):**

3425 Data Type: `OperationalStatus`. The `OperationalStatus` before the  
3426 transition.

3427

3428 **Possible Error Conditions:**

3429 None.

### 3430 **6.9.2 TTOperStatusAlarm.getState**

3431 Queries the `TTOperStatusAlarm` for the value of the `OperationalStatus` attribute  
3432 after the transition.

3433

3434 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3435 implement this command.

3436

3437 **Usage:**

3438 `TTOperStatusAlarm.getState (void): OperationalStatus`

3439

3440 **Parameter(s):**

3441 Data Type: `void`. This command takes no parameters.

3442

3443 **Return Value(s):**

3444 Data Type: `OperationalStatus`. The `OperationalStatus` after the  
3445 transition.

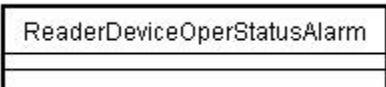
3446

3447 **Possible Error Conditions:**

3448 None.

### 3449 **6.10 ReaderDeviceOperStatusAlarm**

3450 `ReaderDeviceOperStatusAlarm` extends the `TTOperStatusAlarm` class. Its  
3451 receipt signals a change in the operational status of a Reader. The abstract model's  
3452 `ReaderDevice.OperStatusAlarmControl` data element controls the triggering of  
3453 alarms of this type.



3454

3455 **Figure 26 ReaderDeviceStatusAlarm UML**

3456

3457 **Compliance Requirement:** Compliant systems SHALL implement this alarm if  
3458 AlarmChannel is implemented.

3459

## 3460 **6.11 IOPortOperStatusAlarm**

3461 IOPortOperStatusAlarm extends the TTOperStatusAlarm class. Its receipt signals  
3462 a change in the operational status of a Reader's IO Port. The abstract model's  
3463 IOPort.OperStatusAlarmControl data element controls the triggering of alarms of  
3464 this type.



3465

3466

3467 **Figure 27 IOPortOperStatusAlarm UML**

3468

3469 The IOPortName message attribute SHALL identify the name of the IO Port that  
3470 experienced the alarm-triggering state transition, i.e., the value of the respective  
3471 IOPort.Name model element.

3472

3473 **Compliance Requirement:** Compliant systems SHALL implement this alarm if  
3474 AlarmChannel and IOPort are implemented.

### 3475 **6.11.1 IOPortOperStatusAlarm.getIOPortName**

3476 Queries the IOPortStatusAlarm for the value of the IOPortName attribute.

3477

3478 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3479 implement this command.

3480

3481 **Usage:**

3482 IOPortOperStatus.getIOPortName (void): string

3483

3484 **Parameter(s):**

3485 Data Type: void. This command takes no parameters.

3486

3487 **Return Value(s):**

3488 Data Type: string. The name of the IOPort.

3489

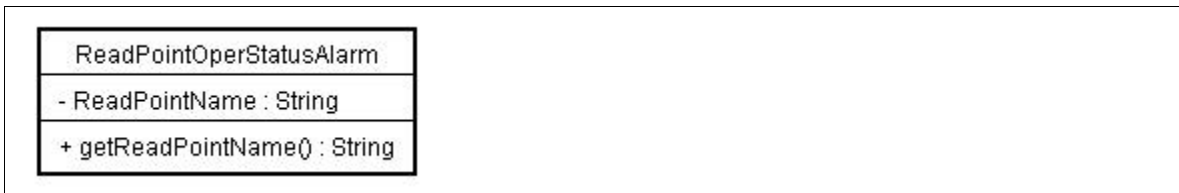
3490 **Possible Error Conditions:**

3491 None.

3492

## 3493 **6.12 ReadPointOperStatusAlarm**

3494 ReadPointOperStatusAlarm extends the TTOperStatusAlarm class. Its receipt  
3495 signals a change in the operational status of one of a Reader's Read Points. The abstract  
3496 model's ReadPoint.OperStatusAlarmControl data element controls the triggering  
3497 of alarms of this type.



3498

3499

3500 **Figure 28 ReadPointOperStatusAlarm UML**

3501

3502 The ReadPointName message attribute SHALL identify the name of the Read Point that  
3503 experienced the alarm-triggering state transition, i.e., the value of the respective  
3504 ReadPoint.Name model element.

3505 **Compliance Requirement:** Compliant systems SHALL implement this alarm if  
3506 AlarmChannel is implemented.

### 3507 **6.12.1 ReadPointOperStatusAlarm.getReadPointName**

3508 Queries the ReadPointOperStatusAlarm for the value of the ReadPointName attribute.

3509

3510 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3511 implement this command.

3512

3513 **Usage:**

3514 ReadPointOperStatusAlarm.getReadPointName (void): string

3515

3516 **Parameter(s):**

3517 Data Type: void. This command takes no parameters.

3518

3519 **Return Value(s):**

3520 Data Type: string. The name of the ReadPoint.

3521

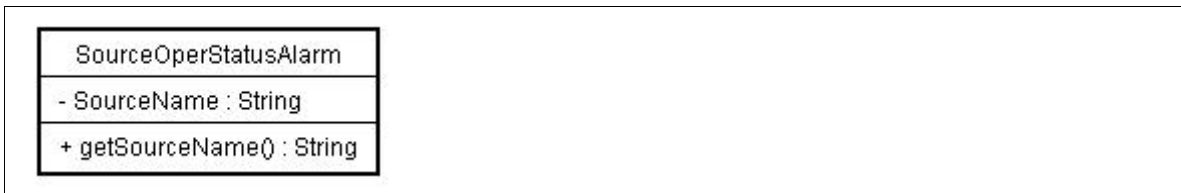
3522 **Possible Error Conditions:**

3523 None.

3524

3525 **6.13 SourceOperStatusAlarm**

3526 SourceOperStatusAlarm extends the TTOperStatusAlarm class. Its receipt signals  
3527 a change in the operational status of a logical source of EPC data on a Reader. The  
3528 abstract model's SourceOperStatusAlarmControl data element controls the  
3529 triggering of alarms of this type.



3530

3531

3532 **Figure 29 SourceOperStatusAlarm UML**

3533

3534 The SourceName message attribute SHALL identify the name of the logical source that  
3535 experienced the alarm-triggering state transition, i.e., the value of the respective  
3536 SourceName model element.

3537 **Compliance Requirement:** Compliant systems SHALL implement this alarm if  
3538 AlarmChannel is implemented.

3539 **6.13.1 SourceOperStatusAlarm.getSourceName**

3540 Queries the SourceOperStatusAlarm for the value of the SourceName attribute.

3541

3542 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL  
3543 implement this command.

3544

3545 **Usage:**

3546 SourceOperStatusAlarm.getSourceName (void): string

3547



3548 **Parameter(s):**

3549 Data Type: void. This command takes no parameters.

3550

3551 **Return Value(s):**

3552 Data Type: string. The name of the Source.

3553

3554 **Possible Error Conditions:**

3555 None.

3556

## 3557 **6.14 NotificationChannelOperStatusAlarm**

3558 NotificationChannelOperStatusAlarm extends the TTOperStatusAlarm class.

3559 Its receipt signals a change in the operational status of one of a Reader's notification

3560 channels. The abstract model's NotificationChannelOperStatusAlarmControl

3561 data element controls the triggering of alarms of this type.



3562

3563 **Figure 30 NotificationChannelOperStatusAlarm UML**

3564

3565

3566 The NotificationChannelName message attribute SHALL identify

3567 the name of the notification channel that experienced the alarm-triggering state

3568 transition, i.e., the value of the respective NotificationChannel.Name model

3569 element.

3570 **Compliance Requirement:** Compliant systems SHALL implement this alarm if

3571 AlarmChannel and NotificationChannel are implemented.

### 3572 **6.14.1 NotificationChannelOperStatusAlarm.getNotificationC** 3573 **hannelName**

3574 Queries the NotificationChannelOperStatusAlarm for the value of the

3575 NotificationChannelName attribute.

3576

3577 **Compliance Requirement:** If this alarm is implemented, compliant systems SHALL

3578 implement this command.

3579

3580 **Usage:**

3581 NotificationChannelOperStatusAlarm.getNotificationChannelName (void): string  
3582

3583

3584 **Parameter(s):**

3585 Data Type: void. This command takes no parameters.

3586

3587 **Return Value(s):**

3588 Data Type: string. The name of the NotificationChannel.

3589

3590 **Possible Error Conditions:**

3591 None.

3592

## 3593 7 Enumerated types

### 3594 7.1.1 AdministrativeStatus

3595 The administrative status represents the host's desired state for an object.

Value	Description
UP	The host desires that the operational status be "up"
DOWN	The host desires that the operational status be "down"

3596

### 3597 7.1.2 OperationalStatus

3598 The operational status represents the actual state for an object.

Value	Description
UNKNOWN	The reader lacks the ability to determine the operational status of the object. The real status may be "up", "down", or "other".
UP	The object is operational. For aggregate objects (i.e. objects that encompass other objects, as a source may encompass multiple read points) this means that all objects that are administratively up are either operationally up or operationally unknown.

DOWN	The object is not operational. For aggregate objects (i.e. objects that encompass other objects, as a source may encompass multiple read points) this means that all objects that are administratively up are either operationally down or operationally unknown.
OTHER	The object is neither fully up, nor fully down. For aggregate objects (i.e. objects that encompass other objects, as a source may encompass multiple read points) this includes the case where all sub-objects are administratively down and the aggregate object is administratively up.
ANY	A wildcard operator. The comparison “ANY”==S is true for all status enumerations, S. This is never returned by the reader, but is useful to hosts describing events. For example a host may desire an alert when the reader state changed from UP to ANY.

3599

### 3600 **7.1.3 EdgeTriggeredAlarmDirection**

3601 Controls the direction in which an edge-triggered alarm should fire.

Value	Description
RISING	Trigger on the rising edge
FALLING	Trigger on the falling edge

3602

### 3603 **7.1.4 EdgeTriggeredAlarmStatus**

3604 Describes the status of an edge-triggered alarm. See the Alarm object description for an  
3605 operational model.

Value	Description
ARMED	Alarm will fire when the monitored value crosses the alarm threshold
FIRED	Alarm has fired; monitored value has not yet crossed re-arm threshold

3606

--

3607

### 3608 **7.1.5 AlarmLevel**

3609 The level of a reported alarm. When the host requests an alarm it defines the alarm level  
3610 (see AlarmControl.setLevel). This additional information is then returned by the reader  
3611 along with each occurrence of that alarm. It is expected that some management  
3612 systems may choose to filter based on the alarm level. The alarm levels are modeled  
3613 after The syslog Protocol IETF draft [SYSLOG].

Value	Description
-------	-------------

EMERGENCY	System is unusable
ALERT	Action must be taken immediately
CRITICAL	Critical conditions
ERROR	Error conditions
WARNING	Warning conditions
NOTICE	Normal but significant conditions
INFORMATIONAL	Informational messages
DEBUG	Debug level messages

3614

--

3615

3616

## 3617 **8 Error Handling**

### 3618 **8.1 Error Conditions**

3619 There are several types of error conditions:

- 3620 • Communication errors, either when a host issues a command or when the reader  
3621 tries to send a notification (e.g., tag list report) to the host
- 3622 • Command execution errors that occur as response to commands sent from the host  
3623 to the reader.

3624 Vendor extensions

3625 This section lists the errors that are unique to the Reader Management Specification.  
3626 Refer to the Reader Protocol Specification 1.1 for the definition of errors inherited from  
3627 the the Reader Protocol Specification 1.1.

### 3628 **8.2 Communication Errors**

#### 3629 **8.2.1 Communication Host-to-Reader**

3630 These kinds of communication errors occur when the host tries to issue a command to the  
3631 reader but the reader is not responding. These conditions are reported by the MTB-  
3632 specific Transport Layer.

#### 3633 **8.2.2 Communication Reader-to-Host**

3634 This specification does not define how the reader reacts when it isn't able to connect or  
3635 exchange data over a notification channel; the exact behavior is implementation

3636 dependent. Notifications could be dropped, sent over a secondary channel, queued for  
3637 later retry, raise a trap in a management console, and/or many other behaviors.

### 3638 **8.3 Command Errors**

3639 All commands support a standard `error` structure, for use in the event of errors. This  
3640 contains standard attributes (some required, some optional) as well as support for vendor  
3641 extensions.

3642 All standard error conditions defined can be reported with additional data about the error  
3643 (formatted and transmitted as per the MTB in use):

- 3644 • **SHALL**: The **error code** and/or **error name**. At least one of these attributes **SHALL**  
3645 be included in the `Error` structure. The exact contents depend on the binding, and  
3646 **SHALL** be constant at the binding level, e.g. MTB X can be defined as always  
3647 including the **error code** (but not the **error name**)
- 3648 • **SHALL** (where applicable): Information about the **error cause**. For example, for  
3649 input parameter errors, information on which parameter failed needs to be included.  
3650 Depending on the binding, this can be either the name or the index of the parameter.  
3651 Vendor extensions are allowed here.
- 3652 • **SHALL** (where applicable): A **vendor name or identifier** for the responses to  
3653 vendor-specific commands **SHALL** be given.
- 3654 • **MAY**: A **descriptive text string**, mainly used for logging purposes. The language for  
3655 the text is implementation-dependent and **MAY** be configurable.

3656 The error code is a 16-bit (2-byte) integer. The following table defines the mapping of  
3657 error conditions / error codes unique to Reader Management. Refer to Reader Protocol  
3658 Specification 1.1 for errors imported from it.

3659

Error Code (Hex)	Error Name	Description
0001	ERROR_UNKNOWN	Refer to Reader Protocol Specification 1.1
0002	ERROR_COMMAND_NOT_SUPPORTED	Refer to Reader Protocol Specification 1.1
0003	ERROR_PARAMETER_INVALID_FORMAT	Refer to Reader Protocol Specification 1.1
0004	ERROR_PARAMETER_MISSING	Refer to Reader Protocol Specification 1.1

Error Code (Hex)	Error Name	Description
0009	ERROR_PARAMETER_LENGTH_EXCEEDED	Refer to Reader Protocol Specification 1.1
000D	ERROR_TRIGGER_NOT_FOUND	Refer to Reader Protocol Specification 1.1
000E	ERROR_READPOINT_NOT_FOUND	Refer to Reader Protocol Specification 1.1
0010	ERROR_SOURCE_NOT_FOUND	Raised when the given Source parameter is not known
1000	ERROR_IOPORT_NOT_FOUND	Raised when the given IOPort parameter is not known
1001	ERROR_ALARM_CHANNEL_NOT_FOUND	Raised when the given AlarmChannel parameter is not known
1002	ERROR_INVALID_ADMIN_STATUS	Raised when the specified Administrative status is not allowed for the given state of the object
1003	ERROR_TOO_MANY_ALARM_CHANNELS	Raised when the command failed due to an internal resource restriction. For instance, not enough memory, maximum number of objects reached, etc.
1004	ERROR_AUTHORIZATION	Raised when the command failed due to lack of rights to perform the operation on the object.

3660

**Table 1: Basic Reader Management Error Codes**

3661

## 3662 **9 Vendor Extensions**

3663 Vendors SHALL use the error conditions defined above for the commands of the  
3664 standard command set. Vendors MAY add additional information on the exact cause of  
3665 the error.

3666 Vendors can extend the standard Reader Management command set. These extension  
3667 commands MAY use the same general error conditions as defined above, however they  
3668 MAY also define additional error conditions. These vendor-defined error conditions  
3669 SHALL use names starting with <VENDOR>\_ERROR..., for example  
3670 ACME\_ERROR\_WHATSIT\_NOT\_ADDED\_NO\_FLABOOZLE if Acme Corp. needed an  
3671 error condition indicating there was no WHATSIT associated with a FLABOOZLE.

3672

3673 Vendor extensions may be:

3674 • Additional commands defined for the following objects:

3675 AlarmChannel  
3676 AlarmControl  
3677 AntennaReadPoint  
3678 EdgeTriggeredAlarmControl  
3679 IOPort  
3680  
3681 NotificationChannel  
3682 ReaderDevice  
3683 ReadPoint  
3684 Source  
3685  
3686 Trigger  
3687 TTOperationalStatusAlarmControl  
3688

3689 • Additional attributes defined for the following objects:

3690 Alarm  
3691  
3692 FailedKillAlarm  
3693 FailedLockAlarm  
3694 FailedMemReadAlarm  
3695  
3696 FailedWriteAlarm  
3697 FreeMemoryAlarm  
3698 IOPortOperStatusAlarm  
3699 NotificationChannelOperStatusAlarm  
3700 OperStatusAlarm  
3701 ReaderDeviceOperStatusAlarm  
3702 ReadPointOperStatusAlarm  
3703 SourceOperStatusAlarm

3704

3705       • Commands to additional objects.

3706

3707       • Additional types of Alarm (i.e. additional subclasses of Alarm).

3708

3709       The exact syntax for expressing vendor extensions is detailed for each Message/Transport  
3710       Binding (MTB).

3711



3712

## 3713 **10 Message/Transport Bindings (MTBs)**

3714 The previous section defined an Abstract Command Set independent of message format  
3715 or communication transport. This section defines the actual mapping of the Command Set  
3716 to the specific Message Formats defined by the Reader Protocol Version 1.0 and SNMP.  
3717 Each message format is independent of the other. A compliant application SHALL  
3718 implement at least one of the Message Formats and MAY implement more than one of  
3719 the message formats. If a particular Message Format is implemented, the compliant  
3720 application SHALL implement all of the operations labeled SHALL. If an operation is  
3721 not supported, the Reader and/ or Host SHALL reply with ERROR\_NOT\_SUPPORTED.

3722 Currently, two message formats are specified, XML, and SNMP. The XML message  
3723 format can be used over any of the 3 transports (Serial, TCP, and HTTP) defined by the  
3724 Reader Protocol Specification 1.1. In the future, additional message formats and  
3725 transports may be defined.

### 3726 **10.1 Address Notation**

3727 All transports support a URI naming convention when an address is required by the  
3728 command set, e.g. `NotificationChannel.setAddress()`. This uses the  
3729 following notation except where explicitly stated otherwise:

3730 `<transport>://<locator>`

3731 For example:

3732 `tcp://foo.bar.com:2000`

3733 `serial://com1:3`

3734 <http://myreaderisawebserver:7183>

3735

3736 Note 1: in the case of a serial connection, where multiple channels have to be multiplexed  
3737 over a single connection, the 'port' is a number that identifies the channel.

3738 Within the SNMP MIB, addresses are defined in compliance with RFC 3291/RFC 4001,  
3739 the INET-ADDRESS MIB. This allows for addressing via IPv4, IPv6, or DNS Fully  
3740 Qualified Domain Name.

### 3741 **10.2 Vendor Extension Details**

3742 Refer to the Reader Protocol 1.1 Vendor Extensions for the syntax used to define new  
3743 Commands, Errors, and Attributes using the XML Message format.

3744 Vendor Extensions for the SNMP binding are implemented through the use of a vendor  
3745 specific MIB. The Vendor specific MIB may reference the EPCglobal Reader  
3746 Management MIB.

3747 The management application needs to know a priori the vendor extensions  
3748 applicable to a reader. The XML schema provided by the reader vendor defines the

3749 operations (standard and extensions). SNMP provides for extensions by defining a  
3750 separate MIB applicable to the reader model/vendor.

3751

## 3752 10.3 XML Message Format

3753 Refer to the Reader Protocol Version 1.1 Specification for the XML Binding rules,  
3754 Object identification Scheme and Data Types. The following sections provide the Reader  
3755 Management specific schema definitions.

### 3756 10.3.1 Command XML Message Encoding (Host-To-Reader)

```
3757 <?xml version="1.0" encoding="UTF-8"?>
3758 <xsd:schema targetNamespace="urn:epcglobal:rm:xsd:1" elementFormDefault="unqualified"
3759 attributeFormDefault="unqualified" version="1.0" xmlns:rm="urn:epcglobal:rm:xsd:1"
3760 xmlns:epcglobal="urn:epcglobal:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3761   <xsd:annotation>
3762     <xsd:documentation xml:lang="en">
3763       <epcglobal:copyright>Copyright&#169;2005-2006 Epcglobal Inc., All Rights Reserved.</epcglobal:copyright>
3764       <epcglobal:disclaimer>EPCglobal Inc., its members, officers, directors, employees, or
3765         agents shall not be liable for any injury, loss, damages, financial or otherwise,
3766         arising from, related to, or caused by the use of this document. The use of said
3767         document shall constitute your express consent to the foregoing exculpation.</epcglobal:disclaimer>
3768       <epcglobal:specification>Reader Management (RM) version 1.0</epcglobal:specification>
3769     </xsd:documentation>
3770   </xsd:annotation>
3771   <xsd:include schemaLocation="RmCommon.xsd"/>
3772   <xsd:import namespace="urn:epcglobal:xsd:1" schemaLocation="./EpcGlobal.xsd"/>
3773   <!-- Reader Management Command -->
3774   <xsd:element name="command">
3775     <xsd:annotation>
3776       <xsd:documentation xml:lang="en"> This element defines a reader management command.
3777       The commands are grouped by the object they belong to. To specify the object a command relates to,
3778       the name of the object is used. Therefore, all commands that return the name of the object
3779       become useless, since the name must be known before it can be queried. All the names of a
3780       particular object type are global irrespective of where they belong to. Hence names need to
3781       be unique within their object type. All the commands are encoded by its straight forward
3782       conversion from UML notation. Each command is wrapped in its object type and contains its
3783       parameters as child elements. </xsd:documentation>
3784     </xsd:annotation>
3785     <xsd:complexType>
3786       <xsd:complexContent>
3787         <xsd:extension base="epcglobal:Document">
3788           <xsd:sequence>
3789             <xsd:element name="id" type="xsd:string">
3790               <xsd:annotation>
3791                 <xsd:documentation xml:lang="en">The id of the command</xsd:documentation>
3792               </xsd:annotation>
3793             </xsd:element>
3794             <xsd:element name="targetName" type="xsd:string" minOccurs="0">
3795               <xsd:annotation>
3796                 <xsd:documentation xml:lang="en">The name of the object on which this command should be
3797                 executed. It can be left out for static methods/ or commands targeted to ReaderDevice object</xsd:documentation>
3798               </xsd:annotation>

```

```

3799     </xsd:element>
3800     <xsd:choice>
3801         <!-- Reader Management (Target) Object Types -->
3802         <!-- elements inside are ordered in alphabetical order -->
3803         <xsd:element name="alarmChannel" type="rm:AlarmChannelCommand">
3804             <xsd:annotation>
3805                 <xsd:documentation xml:lang="en">An Alarm channel</xsd:documentation>
3806             </xsd:annotation>
3807         </xsd:element>
3808         <xsd:element name="alarmControl" type="rm:AlarmControlCommand">
3809             <xsd:annotation>
3810                 <xsd:documentation xml:lang="en">AlarmControl is the base class for all of classes
3811 within the RMP object model responsible for controlling the generation of alarm messages within an EPCglobal
3812 compliant Reader.</xsd:documentation>
3813             </xsd:annotation>
3814         </xsd:element>
3815         <xsd:element name="antennaReadPoint" type="rm:AntennaReadPointCommand">
3816             <xsd:annotation>
3817                 <xsd:documentation xml:lang="en">Extension of Read point object for
3818 antenna.</xsd:documentation>
3819             </xsd:annotation>
3820         </xsd:element>
3821         <xsd:element name="edgeTriggeredAlarmControl" type="rm:EdgeTriggeredAlarmControlCommand">
3822             <xsd:annotation>
3823                 <xsd:documentation xml:lang="en">This class extends AlarmControl to control alarms
3824 generated when a monitored, integer-valued, model element first crosses a threshold value (the
3825 AlarmThreshold).</xsd:documentation>
3826             </xsd:annotation>
3827         </xsd:element>
3828         <xsd:element name="iOPort" type="rm:IOPortCommand">
3829             <xsd:annotation>
3830                 <xsd:documentation xml:lang="en">The hardware element that provides external input
3831 and output lines to connect to other components outside the reader device.</xsd:documentation>
3832             </xsd:annotation>
3833         </xsd:element>
3834         <xsd:element name="notificationChannel" type="rm:NotificationChannelCommand">
3835             <xsd:annotation>
3836                 <xsd:documentation xml:lang="en">The notification channel carries messages
3837 issued asynchronously by the Reader to the Host.</xsd:documentation>
3838             </xsd:annotation>
3839         </xsd:element>
3840         <xsd:element name="readerDevice" type="rm:ReaderDeviceCommand">
3841             <xsd:annotation>
3842                 <xsd:documentation xml:lang="en">A Reader</xsd:documentation>
3843             </xsd:annotation>
3844         </xsd:element>
3845         <xsd:element name="readPoint" type="rm:ReadPointCommand">
3846             <xsd:annotation>
3847                 <xsd:documentation xml:lang="en">A ReadPoint</xsd:documentation>
3848             </xsd:annotation>
3849         </xsd:element>
3850         <xsd:element name="source" type="rm:SourceCommand">
3851             <xsd:annotation>
3852                 <xsd:documentation xml:lang="en">A Read source</xsd:documentation>
3853             </xsd:annotation>
3854         </xsd:element>

```

```

3855     <xsd:element name="trigger" type="rm:TriggerCommand">
3856         <xsd:annotation>
3857             <xsd:documentation xml:lang="en">A Read/Notify Trigger</xsd:documentation>
3858         </xsd:annotation>
3859     </xsd:element>
3860     <xsd:element name="tOperationalStatusAlarmControl"
3861 type="rm:TOperationalStatusAlarmControlCommand">
3862         <xsd:annotation>
3863             <xsd:documentation xml:lang="en">This class extends AlarmControl to control alarms
3864 generated when a monitored model element of type OperationalStatus transitions to a new
3865 value.</xsd:documentation>
3866         </xsd:annotation>
3867     </xsd:element>
3868     <xsd:any namespace="##any" processContents="lax">
3869         <xsd:annotation>
3870             <xsd:documentation>For standard and vendor extensions</xsd:documentation>
3871         </xsd:annotation>
3872     </xsd:any>
3873 </xsd:choice>
3874 </xsd:sequence>
3875 </xsd:extension>
3876 </xsd:complexContent>
3877 </xsd:complexType>
3878 </xsd:element>
3879 <!-- Reader Management (Target) Object Types -->
3880 <!-- types are ordered in alphabetical order -->
3881 <!-- inside each type commands are listed in the order in which it is defined in the RM specification. -->
3882 <xsd:complexType name="AlarmChannelCommand">
3883     <xsd:choice>
3884         <xsd:annotation>
3885             <xsd:documentation xml:lang="en"> Alarm channel object commands.</xsd:documentation>
3886         </xsd:annotation>
3887         <xsd:element name="create">
3888             <xsd:annotation>
3889                 <xsd:documentation xml:lang="en">static method to create an Alarm channel. </xsd:documentation>
3890             </xsd:annotation>
3891             <xsd:complexType>
3892                 <xsd:sequence>
3893                     <xsd:element name="name" type="xsd:string">
3894                         <xsd:annotation>
3895                             <xsd:documentation xml:lang="en"> Alarm channel name </xsd:documentation>
3896                         </xsd:annotation>
3897                     </xsd:element>
3898                     <xsd:element name="addr" type="rm:AddressParamType">
3899                         <xsd:annotation>
3900                             <xsd:documentation xml:lang="en"> The (host) address of where alarms will be sent
3901 to.</xsd:documentation>
3902                         </xsd:annotation>
3903                     </xsd:element>
3904                 </xsd:sequence>
3905             </xsd:complexType>
3906         </xsd:element>
3907         <xsd:element name="getName">
3908             <xsd:annotation>
3909                 <xsd:documentation xml:lang="en">get the alarm channel name</xsd:documentation>
3910             </xsd:annotation>

```

```

3911     <xsd:complexType/>
3912 </xsd:element>
3913 <xsd:element name="getAddress">
3914     <xsd:annotation>
3915         <xsd:documentation xml:lang="en">get the (host) address to which this AlarmChannel object sends its
3916 alarms.</xsd:documentation>
3917     </xsd:annotation>
3918     <xsd:complexType/>
3919 </xsd:element>
3920 <xsd:element name="setAddress">
3921     <xsd:annotation>
3922         <xsd:documentation xml:lang="en">set the (host) address to which this AlarmChannel object sends its
3923 alarms.</xsd:documentation>
3924     </xsd:annotation>
3925     <xsd:complexType>
3926         <xsd:sequence>
3927             <xsd:element name="addr" type="rm:AddressParamType">
3928                 <xsd:annotation>
3929                     <xsd:documentation xml:lang="en"> address to set.</xsd:documentation>
3930                 </xsd:annotation>
3931             </xsd:element>
3932         </xsd:sequence>
3933     </xsd:complexType>
3934 </xsd:element>
3935 <xsd:any namespace="##any" processContents="lax">
3936     <xsd:annotation>
3937         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
3938     </xsd:annotation>
3939 </xsd:any>
3940 </xsd:choice>
3941 </xsd:complexType>
3942 <xsd:complexType name="AlarmControlCommand">
3943     <xsd:choice>
3944         <xsd:annotation>
3945             <xsd:documentation xml:lang="en"> Alarm control object commands. </xsd:documentation>
3946         </xsd:annotation>
3947         <xsd:element name="getName">
3948             <xsd:annotation>
3949                 <xsd:documentation xml:lang="en">get the alarm control name</xsd:documentation>
3950             </xsd:annotation>
3951             <xsd:complexType/>
3952         </xsd:element>
3953         <xsd:element name="getEnabled">
3954             <xsd:annotation>
3955                 <xsd:documentation xml:lang="en">query the reader for the current value of the AlarmControlâ€™s
3956 Enabled attribute.</xsd:documentation>
3957             </xsd:annotation>
3958             <xsd:complexType/>
3959         </xsd:element>
3960         <xsd:element name="setEnabled">
3961             <xsd:annotation>
3962                 <xsd:documentation xml:lang="en">enable or disable alarm generation.</xsd:documentation>
3963             </xsd:annotation>
3964             <xsd:complexType>
3965                 <xsd:sequence>
3966                     <xsd:element name="enable" type="xsd:boolean">

```

```

3967         <xsd:annotation>
3968             <xsd:documentation xml:lang="en">whether to enable the alarm generation.</xsd:documentation>
3969         </xsd:annotation>
3970     </xsd:element>
3971 </xsd:sequence>
3972 </xsd:complexType>
3973 </xsd:element>
3974 <xsd:element name="getLevel">
3975     <xsd:annotation>
3976         <xsd:documentation xml:lang="en">query the reader for the current value of the AlarmControlâ€™s Level
3977 attribute.</xsd:documentation>
3978     </xsd:annotation>
3979 </xsd:complexType/>
3980 </xsd:element>
3981 <xsd:element name="setLevel">
3982     <xsd:annotation>
3983         <xsd:documentation xml:lang="en">set the Level attribute.</xsd:documentation>
3984     </xsd:annotation>
3985 </xsd:complexType>
3986     <xsd:sequence>
3987         <xsd:element name="alarmLevel" type="rm:AlarmLevelParamType">
3988             <xsd:annotation>
3989                 <xsd:documentation xml:lang="en">Alarm level to set</xsd:documentation>
3990             </xsd:annotation>
3991         </xsd:element>
3992     </xsd:sequence>
3993 </xsd:complexType>
3994 </xsd:element>
3995 <xsd:element name="getSuppressInterval">
3996     <xsd:annotation>
3997         <xsd:documentation xml:lang="en">query the reader for the current value of the AlarmControlâ€™s
3998 SuppressInterval attribute.</xsd:documentation>
3999     </xsd:annotation>
4000 </xsd:complexType/>
4001 </xsd:element>
4002 <xsd:element name="setSuppressInterval">
4003     <xsd:annotation>
4004         <xsd:documentation xml:lang="en">set the SuppressInterval attribute.</xsd:documentation>
4005     </xsd:annotation>
4006 </xsd:complexType>
4007     <xsd:sequence>
4008         <xsd:element name="suppressInterval" type="xsd:int">
4009             <xsd:annotation>
4010                 <xsd:documentation xml:lang="en">SuppressInterval to set in seconds</xsd:documentation>
4011             </xsd:annotation>
4012         </xsd:element>
4013     </xsd:sequence>
4014 </xsd:complexType>
4015 </xsd:element>
4016 <xsd:any namespace="##any" processContents="lax">
4017     <xsd:annotation>
4018         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4019     </xsd:annotation>
4020 </xsd:any>
4021 </xsd:choice>
4022 </xsd:complexType>

```

```

4023 <xsd:complexType name="AntennaReadPointCommand">
4024   <xsd:choice>
4025     <xsd:annotation>
4026       <xsd:documentation xml:lang="en"> Antenna read point object commands </xsd:documentation>
4027     </xsd:annotation>
4028     <xsd:element name="getIdentificationCount">
4029       <xsd:annotation>
4030         <xsd:documentation xml:lang="en">get the number of the successful tags that have been
4031           identified across an AntennaReadPoint. </xsd:documentation>
4032       </xsd:annotation>
4033       <xsd:complexType/>
4034     </xsd:element>
4035     <xsd:element name="getFailedIdentificationCount">
4036       <xsd:annotation>
4037         <xsd:documentation xml:lang="en">get the number of the failed tag identification attempts at the
4038           AntennaReadPoint. </xsd:documentation>
4039       </xsd:annotation>
4040       <xsd:complexType/>
4041     </xsd:element>
4042     <xsd:element name="getMemReadCount">
4043       <xsd:annotation>
4044         <xsd:documentation xml:lang="en">get the number of tag memory reads at the AntennaReadPoint.
4045       </xsd:documentation>
4046       </xsd:annotation>
4047       <xsd:complexType/>
4048     </xsd:element>
4049     <xsd:element name="getFailedMemReadCount">
4050       <xsd:annotation>
4051         <xsd:documentation xml:lang="en">get the number of the failed tag memory reads at the
4052           AntennaReadPoint. </xsd:documentation>
4053       </xsd:annotation>
4054       <xsd:complexType/>
4055     </xsd:element>
4056     <xsd:element name="getFailedMemReadAlarmControl">
4057       <xsd:annotation>
4058         <xsd:documentation xml:lang="en">get the AntennaReadPoint's failed memory read alarm
4059           control.</xsd:documentation>
4060       </xsd:annotation>
4061       <xsd:complexType/>
4062     </xsd:element>
4063     <xsd:element name="getWriteCount">
4064       <xsd:annotation>
4065         <xsd:documentation xml:lang="en">get the number of successful tag writes at the AntennaReadPoint.
4066       </xsd:documentation>
4067       </xsd:annotation>
4068       <xsd:complexType/>
4069     </xsd:element>
4070     <xsd:element name="getFailedWriteCount">
4071       <xsd:annotation>
4072         <xsd:documentation xml:lang="en">get the number of the failed attempts to write tags at the
4073           AntennaReadPoint. </xsd:documentation>
4074       </xsd:annotation>
4075       <xsd:complexType/>
4076     </xsd:element>
4077     <xsd:element name="getFailedWriteAlarmControl">
4078       <xsd:annotation>

```

```

4079         <xsd:documentation xml:lang="en">get the AntennaReadPoint's failed write alarm
4080 control.</xsd:documentation>
4081         </xsd:annotation>
4082         <xsd:complexType/>
4083     </xsd:element>
4084     <xsd:element name="getKillCount">
4085         <xsd:annotation>
4086             <xsd:documentation xml:lang="en">get the number of tags successfully killed at the AntennaReadPoint.
4087 </xsd:documentation>
4088         </xsd:annotation>
4089         <xsd:complexType/>
4090     </xsd:element>
4091     <xsd:element name="getFailedKillCount">
4092         <xsd:annotation>
4093             <xsd:documentation xml:lang="en">get the number of the failed tag kills at the AntennaReadPoint.
4094 </xsd:documentation>
4095         </xsd:annotation>
4096         <xsd:complexType/>
4097     </xsd:element>
4098     <xsd:element name="getFailedKillAlarmControl">
4099         <xsd:annotation>
4100             <xsd:documentation xml:lang="en">get the AntennaReadPoint's failed kill alarm
4101 control.</xsd:documentation>
4102         </xsd:annotation>
4103         <xsd:complexType/>
4104     </xsd:element>
4105     <xsd:element name="getEraseCount">
4106         <xsd:annotation>
4107             <xsd:documentation xml:lang="en">get the number of tags successfully erased at the
4108 AntennaReadPoint.</xsd:documentation>
4109         </xsd:annotation>
4110         <xsd:complexType/>
4111     </xsd:element>
4112     <xsd:element name="getFailedEraseCount">
4113         <xsd:annotation>
4114             <xsd:documentation xml:lang="en">get the number of the failed tag erasures at the AntennaReadPoint.
4115 </xsd:documentation>
4116         </xsd:annotation>
4117         <xsd:complexType/>
4118     </xsd:element>
4119     <xsd:element name="getFailedEraseAlarmControl">
4120         <xsd:annotation>
4121             <xsd:documentation xml:lang="en">get the AntennaReadPoint's failed erase alarm
4122 control.</xsd:documentation>
4123         </xsd:annotation>
4124         <xsd:complexType/>
4125     </xsd:element>
4126     <xsd:element name="getLockCount">
4127         <xsd:annotation>
4128             <xsd:documentation xml:lang="en">get the number of tags successfully locked at the AntennaReadPoint.
4129 </xsd:documentation>
4130         </xsd:annotation>
4131         <xsd:complexType/>
4132     </xsd:element>
4133     <xsd:element name="getFailedLockCount">
4134         <xsd:annotation>

```



```

4135         <xsd:documentation xml:lang="en">get the number of the failed tag locks at the AntennaReadPoint.
4136     </xsd:documentation>
4137         </xsd:annotation>
4138         <xsd:complexType/>
4139     </xsd:element>
4140     <xsd:element name="getFailedLockAlarmControl">
4141         <xsd:annotation>
4142             <xsd:documentation xml:lang="en">get the AntennaReadPoint's failed lock alarm
4143 control.</xsd:documentation>
4144         </xsd:annotation>
4145         <xsd:complexType/>
4146     </xsd:element>
4147     <xsd:element name="getTimeEnergized">
4148         <xsd:annotation>
4149             <xsd:documentation xml:lang="en">get the number of milliseconds the AntennaReadPoint has been
4150 energized in order to communicate with tags.</xsd:documentation>
4151         </xsd:annotation>
4152         <xsd:complexType/>
4153     </xsd:element>
4154     <xsd:element name="getPowerLevel">
4155         <xsd:annotation>
4156             <xsd:documentation xml:lang="en">get the current transmit power level.</xsd:documentation>
4157         </xsd:annotation>
4158         <xsd:complexType/>
4159     </xsd:element>
4160     <xsd:element name="getNoiseLevel">
4161         <xsd:annotation>
4162             <xsd:documentation xml:lang="en">get the current noise level.</xsd:documentation>
4163         </xsd:annotation>
4164         <xsd:complexType/>
4165     </xsd:element>
4166     <xsd:any namespace="##any" processContents="lax">
4167         <xsd:annotation>
4168             <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4169         </xsd:annotation>
4170     </xsd:any>
4171 </xsd:choice>
4172 </xsd:complexType>
4173 <xsd:complexType name="EdgeTriggeredAlarmControlCommand">
4174     <xsd:choice>
4175         <xsd:annotation>
4176             <xsd:documentation xml:lang="en"> Edge triggered alarm control object commands. </xsd:documentation>
4177         </xsd:annotation>
4178         <xsd:element name="getAlarmThreshold">
4179             <xsd:annotation>
4180                 <xsd:documentation xml:lang="en">query the reader for the current value of the
4181 EdgeTriggeredAlarmControl™s AlarmThreshold attribute.</xsd:documentation>
4182             </xsd:annotation>
4183             <xsd:complexType/>
4184         </xsd:element>
4185         <xsd:element name="setAlarmThreshold">
4186             <xsd:annotation>
4187                 <xsd:documentation xml:lang="en">set the current value of the EdgeTriggeredAlarmControl™s
4188 AlarmThreshold attribute.</xsd:documentation>
4189             </xsd:annotation>
4190         <xsd:complexType/>

```

```

4191     <xsd:sequence>
4192         <xsd:element name="alarmThreshold" type="xsd:int">
4193             <xsd:annotation>
4194                 <xsd:documentation xml:lang="en">threshold to set</xsd:documentation>
4195             </xsd:annotation>
4196         </xsd:element>
4197     </xsd:sequence>
4198 </xsd:complexType>
4199 </xsd:element>
4200 <xsd:element name="getRearmThreshold">
4201     <xsd:annotation>
4202         <xsd:documentation xml:lang="en">query the reader for the current value of the
4203 EdgeTriggeredAlarmControlâ€™s RearmThreshold attribute.</xsd:documentation>
4204     </xsd:annotation>
4205     <xsd:complexType/>
4206 </xsd:element>
4207 <xsd:element name="setRearmThreshold">
4208     <xsd:annotation>
4209         <xsd:documentation xml:lang="en">set the current value of the RearmThreshold
4210 attribute.</xsd:documentation>
4211     </xsd:annotation>
4212     <xsd:complexType>
4213         <xsd:sequence>
4214             <xsd:element name="rearmThreshold" type="xsd:int">
4215                 <xsd:annotation>
4216                     <xsd:documentation xml:lang="en">threshold to set</xsd:documentation>
4217                 </xsd:annotation>
4218             </xsd:element>
4219         </xsd:sequence>
4220     </xsd:complexType>
4221 </xsd:element>
4222 <xsd:element name="getDirection">
4223     <xsd:annotation>
4224         <xsd:documentation xml:lang="en">query the reader for the current value of the
4225 EdgeTriggeredAlarmControlâ€™s Direction attribute.</xsd:documentation>
4226     </xsd:annotation>
4227     <xsd:complexType/>
4228 </xsd:element>
4229 <xsd:element name="setDirection">
4230     <xsd:annotation>
4231         <xsd:documentation xml:lang="en">set the current value of the EdgeTriggeredAlarmControlâ€™s Direction
4232 attribute.</xsd:documentation>
4233     </xsd:annotation>
4234     <xsd:complexType>
4235         <xsd:sequence>
4236             <xsd:element name="direction" type="rm:EdgeTriggeredAlarmDirectionParamType">
4237                 <xsd:annotation>
4238                     <xsd:documentation xml:lang="en">direction to set</xsd:documentation>
4239                 </xsd:annotation>
4240             </xsd:element>
4241         </xsd:sequence>
4242     </xsd:complexType>
4243 </xsd:element>
4244 <xsd:element name="getStatus">
4245     <xsd:annotation>

```

```

4246         <xsd:documentation xml:lang="en">query the reader for the current value of the
4247 EdgeTriggeredAlarmControlâ€™s Status attribute.</xsd:documentation>
4248     </xsd:annotation>
4249     <xsd:complexType/>
4250 </xsd:element>
4251 <xsd:any namespace="##any" processContents="lax">
4252     <xsd:annotation>
4253         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4254     </xsd:annotation>
4255 </xsd:any>
4256 </xsd:choice>
4257 </xsd:complexType>
4258 <xsd:complexType name="IOPortCommand">
4259     <xsd:choice>
4260         <xsd:annotation>
4261             <xsd:documentation xml:lang="en">IO port object commands. </xsd:documentation>
4262         </xsd:annotation>
4263         <xsd:element name="getName">
4264             <xsd:annotation>
4265                 <xsd:documentation xml:lang="en">get the IO port name</xsd:documentation>
4266             </xsd:annotation>
4267             <xsd:complexType/>
4268         </xsd:element>
4269         <xsd:element name="getDescription">
4270             <xsd:annotation>
4271                 <xsd:documentation xml:lang="en">get a textual description of the IO-port.</xsd:documentation>
4272             </xsd:annotation>
4273             <xsd:complexType/>
4274         </xsd:element>
4275         <xsd:element name="setDescription">
4276             <xsd:annotation>
4277                 <xsd:documentation xml:lang="en">set (associate) a textual description with an IO-
4278 port.</xsd:documentation>
4279             </xsd:annotation>
4280             <xsd:complexType>
4281                 <xsd:sequence>
4282                     <xsd:element name="description" type="xsd:string">
4283                         <xsd:annotation>
4284                             <xsd:documentation xml:lang="en">description to set</xsd:documentation>
4285                         </xsd:annotation>
4286                     </xsd:element>
4287                 </xsd:sequence>
4288             </xsd:complexType>
4289         </xsd:element>
4290         <xsd:element name="getOperStatus">
4291             <xsd:annotation>
4292                 <xsd:documentation xml:lang="en">query for the operational status of an IO-port.</xsd:documentation>
4293             </xsd:annotation>
4294             <xsd:complexType/>
4295         </xsd:element>
4296         <xsd:element name="getAdminStatus">
4297             <xsd:annotation>
4298                 <xsd:documentation xml:lang="en">query for the administrative status of an IO-port.</xsd:documentation>
4299             </xsd:annotation>
4300             <xsd:complexType/>
4301         </xsd:element>

```

```

4302 <xsd:element name="setAdminStatus">
4303   <xsd:annotation>
4304     <xsd:documentation xml:lang="en">set the administrative status of an IO-port.</xsd:documentation>
4305   </xsd:annotation>
4306   <xsd:complexType>
4307     <xsd:sequence>
4308       <xsd:element name="administrativeStatus" type="rm:AdministrativeStatusParamType">
4309         <xsd:annotation>
4310           <xsd:documentation xml:lang="en">Administrative status to set</xsd:documentation>
4311         </xsd:annotation>
4312       </xsd:element>
4313     </xsd:sequence>
4314   </xsd:complexType>
4315 </xsd:element>
4316 <xsd:element name="getOperStatusAlarmControl">
4317   <xsd:annotation>
4318     <xsd:documentation xml:lang="en">get the IOPort's operational status alarm
4319 control.</xsd:documentation>
4320   </xsd:annotation>
4321   <xsd:complexType/>
4322 </xsd:element>
4323 <xsd:any namespace="##any" processContents="lax">
4324   <xsd:annotation>
4325     <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4326   </xsd:annotation>
4327 </xsd:any>
4328 </xsd:choice>
4329 </xsd:complexType>
4330 <xsd:complexType name="NotificationChannelCommand">
4331   <xsd:choice>
4332     <xsd:annotation>
4333       <xsd:documentation xml:lang="en">Notification channel object commands</xsd:documentation>
4334     </xsd:annotation>
4335     <xsd:element name="getLastNotificationAttempt">
4336       <xsd:annotation>
4337         <xsd:documentation xml:lang="en">get the timestamp (TimeTicks) when the last attempt was made to
4338 send a notification to the given address.</xsd:documentation>
4339       </xsd:annotation>
4340       <xsd:complexType/>
4341     </xsd:element>
4342     <xsd:element name="getLastSuccessfulNotification">
4343       <xsd:annotation>
4344         <xsd:documentation xml:lang="en">get the timestamp (TimeTicks) when the last successful notification
4345 was send to the given address.</xsd:documentation>
4346       </xsd:annotation>
4347       <xsd:complexType/>
4348     </xsd:element>
4349   </xsd:choice>
4350 </xsd:complexType>
4351 <xsd:element name="getOperStatus">
4352   <xsd:annotation>
4353     <xsd:documentation xml:lang="en">query the NotificationChannel for its OperationalStatus
4354 object.</xsd:documentation>
4355   </xsd:annotation>
4356   <xsd:complexType/>
4357 </xsd:element>

```

```

4358         <xsd:documentation xml:lang="en">set the administrative status of Notification
4359 channel.</xsd:documentation>
4360     </xsd:annotation>
4361     <xsd:complexType>
4362     <xsd:sequence>
4363         <xsd:element name="administrativeStatus" type="rm:AdministrativeStatusParamType">
4364             <xsd:annotation>
4365                 <xsd:documentation xml:lang="en"> administrative status to set. </xsd:documentation>
4366             </xsd:annotation>
4367         </xsd:element>
4368     </xsd:sequence>
4369     </xsd:complexType>
4370 </xsd:element>
4371 <xsd:element name="getAdminStatus">
4372     <xsd:annotation>
4373         <xsd:documentation xml:lang="en">query the NotificationChannel for its administrative
4374 status.</xsd:documentation>
4375     </xsd:annotation>
4376     <xsd:complexType/>
4377 </xsd:element>
4378 <xsd:element name="getOperStatusAlarmControl">
4379     <xsd:annotation>
4380         <xsd:documentation xml:lang="en">query the notification channel for its operational status alarm
4381 control.</xsd:documentation>
4382     </xsd:annotation>
4383     <xsd:complexType/>
4384 </xsd:element>
4385 <xsd:any namespace="##any" processContents="lax">
4386     <xsd:annotation>
4387         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4388     </xsd:annotation>
4389 </xsd:any>
4390 </xsd:choice>
4391 </xsd:complexType>
4392 <xsd:complexType name="ReaderDeviceCommand">
4393     <xsd:annotation>
4394         <xsd:documentation xml:lang="en">Reader Device object commands. </xsd:documentation>
4395     </xsd:annotation>
4396     <xsd:choice>
4397         <xsd:element name="getDescription">
4398             <xsd:annotation>
4399                 <xsd:documentation xml:lang="en">query the Reader for its user defined
4400 description.</xsd:documentation>
4401             </xsd:annotation>
4402         <xsd:complexType/>
4403     </xsd:element>
4404     <xsd:element name="setDescription">
4405         <xsd:annotation>
4406             <xsd:documentation xml:lang="en">set the reader's user-defined description.</xsd:documentation>
4407         </xsd:annotation>
4408     <xsd:complexType>
4409     <xsd:sequence>
4410         <xsd:element name="description" type="xsd:string">
4411             <xsd:annotation>
4412                 <xsd:documentation xml:lang="en"> description to set</xsd:documentation>
4413             </xsd:annotation>

```

```

4414         </xsd:element>
4415     </xsd:sequence>
4416 </xsd:complexType>
4417 </xsd:element>
4418 <xsd:element name="getLocationDescription">
4419     <xsd:annotation>
4420         <xsd:documentation xml:lang="en">query the reader for its user defined location
4421 description.</xsd:documentation>
4422     </xsd:annotation>
4423 </xsd:complexType/>
4424 </xsd:element>
4425 <xsd:element name="setLocationDescription">
4426     <xsd:annotation>
4427         <xsd:documentation xml:lang="en">set the reader's user-defined location
4428 description.</xsd:documentation>
4429     </xsd:annotation>
4430 </xsd:complexType>
4431 <xsd:sequence>
4432     <xsd:element name="locationDescription" type="xsd:string">
4433         <xsd:annotation>
4434             <xsd:documentation xml:lang="en"> location description to set. </xsd:documentation>
4435         </xsd:annotation>
4436     </xsd:element>
4437 </xsd:sequence>
4438 </xsd:complexType>
4439 </xsd:element>
4440 <xsd:element name="getContact">
4441     <xsd:annotation>
4442         <xsd:documentation xml:lang="en">query the reader for its user-defined contact
4443 description.</xsd:documentation>
4444     </xsd:annotation>
4445 </xsd:complexType/>
4446 </xsd:element>
4447 <xsd:element name="setContact">
4448     <xsd:annotation>
4449         <xsd:documentation xml:lang="en">set the reader's user-defined contact
4450 description.</xsd:documentation>
4451     </xsd:annotation>
4452 </xsd:complexType>
4453 <xsd:sequence>
4454     <xsd:element name="contact" type="xsd:string">
4455         <xsd:annotation>
4456             <xsd:documentation xml:lang="en"> contact to set</xsd:documentation>
4457         </xsd:annotation>
4458     </xsd:element>
4459 </xsd:sequence>
4460 </xsd:complexType>
4461 </xsd:element>
4462 <xsd:element name="getSerialNumber">
4463     <xsd:annotation>
4464         <xsd:documentation xml:lang="en">query the reader for its serial number.</xsd:documentation>
4465     </xsd:annotation>
4466 </xsd:complexType/>
4467 </xsd:element>
4468 <xsd:element name="getOperStatus">
4469     <xsd:annotation>

```

```

4470     <xsd:documentation xml:lang="en">query the reader for its OperationalStatus
4471 object.</xsd:documentation>
4472     </xsd:annotation>
4473     <xsd:complexType/>
4474 </xsd:element>
4475 <xsd:element name="getOperStatusAlarmControl">
4476     <xsd:annotation>
4477         <xsd:documentation xml:lang="en">query the reader for its operational status alarm
4478 control.</xsd:documentation>
4479     </xsd:annotation>
4480     <xsd:complexType/>
4481 </xsd:element>
4482 <xsd:element name="getFreeMemory">
4483     <xsd:annotation>
4484         <xsd:documentation xml:lang="en">query the reader for its available free memory.</xsd:documentation>
4485     </xsd:annotation>
4486     <xsd:complexType/>
4487 </xsd:element>
4488 <xsd:element name="getFreeMemoryAlarmControl">
4489     <xsd:annotation>
4490         <xsd:documentation xml:lang="en">query the reader for its FreeMemoryAlarmControl
4491 object.</xsd:documentation>
4492     </xsd:annotation>
4493     <xsd:complexType/>
4494 </xsd:element>
4495 <xsd:element name="getNTPServers">
4496     <xsd:annotation>
4497         <xsd:documentation xml:lang="en">query the reader for a list of NTP servers used by it to synchronize its
4498 current UTC clock (TimeUTC).</xsd:documentation>
4499     </xsd:annotation>
4500     <xsd:complexType/>
4501 </xsd:element>
4502 <xsd:element name="getDHCPserver">
4503     <xsd:annotation>
4504         <xsd:documentation xml:lang="en">query the reader for the DHCP server currently used by the device for
4505 DHCP requests.</xsd:documentation>
4506     </xsd:annotation>
4507     <xsd:complexType/>
4508 </xsd:element>
4509 <xsd:element name="getIOPort">
4510     <xsd:annotation>
4511         <xsd:documentation xml:lang="en">get the IO Port with the specified name currently associated with this
4512 Reader.</xsd:documentation>
4513     </xsd:annotation>
4514     <xsd:complexType>
4515         <xsd:sequence>
4516             <xsd:element name="name" type="xsd:string">
4517                 <xsd:annotation>
4518                     <xsd:documentation xml:lang="en">IO port name </xsd:documentation>
4519                 </xsd:annotation>
4520             </xsd:element>
4521         </xsd:sequence>
4522     </xsd:complexType>
4523 </xsd:element>
4524 <xsd:element name="getAllIOPorts">
4525     <xsd:annotation>

```

```

4526         <xsd:documentation xml:lang="en">query the reader for all its IOPort objects.</xsd:documentation>
4527     </xsd:annotation>
4528     <xsd:complexType/>
4529 </xsd:element>
4530 <xsd:element name="resetStatistics">
4531     <xsd:annotation>
4532         <xsd:documentation xml:lang="en">reset reader's entire internal statistic counters to
4533 zero.</xsd:documentation>
4534     </xsd:annotation>
4535     <xsd:complexType/>
4536 </xsd:element>
4537 <xsd:element name="removeAlarmChannels">
4538     <xsd:annotation>
4539         <xsd:documentation xml:lang="en">remove the specified AlarmChannels from the list of AlarmChannels
4540 currently associated with the reader.</xsd:documentation>
4541     </xsd:annotation>
4542     <xsd:complexType>
4543         <xsd:sequence>
4544             <xsd:element name="channels" type="rm:AlarmChannelListParamType">
4545                 <xsd:annotation>
4546                     <xsd:documentation xml:lang="en"> alarm channels to remove. </xsd:documentation>
4547                 </xsd:annotation>
4548             </xsd:element>
4549         </xsd:sequence>
4550     </xsd:complexType>
4551 </xsd:element>
4552 <xsd:element name="removeAllAlarmChannels">
4553     <xsd:annotation>
4554         <xsd:documentation xml:lang="en">remove all AlarmChannels currently associated with the
4555 reader.</xsd:documentation>
4556     </xsd:annotation>
4557     <xsd:complexType/>
4558 </xsd:element>
4559 <xsd:element name="getAlarmChannel">
4560     <xsd:annotation>
4561         <xsd:documentation xml:lang="en">get the AlarmChannel with the specified name currently associated
4562 with the reader.</xsd:documentation>
4563     </xsd:annotation>
4564     <xsd:complexType>
4565         <xsd:sequence>
4566             <xsd:element name="name" type="xsd:string">
4567                 <xsd:annotation>
4568                     <xsd:documentation xml:lang="en"> Alarm channel name </xsd:documentation>
4569                 </xsd:annotation>
4570             </xsd:element>
4571         </xsd:sequence>
4572     </xsd:complexType>
4573 </xsd:element>
4574 <xsd:element name="getAllAlarmChannels">
4575     <xsd:annotation>
4576         <xsd:documentation xml:lang="en">get all AlarmChannels currently associated with the
4577 reader.</xsd:documentation>
4578     </xsd:annotation>
4579     <xsd:complexType/>
4580 </xsd:element>
4581 <xsd:any namespace="##any" processContents="lax">

```



```

4582     <xsd:annotation>
4583         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4584     </xsd:annotation>
4585 </xsd:any>
4586 </xsd:choice>
4587 </xsd:complexType>
4588 <xsd:complexType name="ReadPointCommand">
4589     <xsd:annotation>
4590         <xsd:documentation xml:lang="en"> Read point object commands.</xsd:documentation>
4591     </xsd:annotation>
4592     <xsd:choice>
4593         <xsd:element name="getClassName">
4594             <xsd:annotation>
4595                 <xsd:documentation xml:lang="en">get the class name of the ReadPoint object. In the current
4596 specification, the only supported read point class is an "AntennaReadPoint".</xsd:documentation>
4597             </xsd:annotation>
4598             <xsd:complexType/>
4599         </xsd:element>
4600         <xsd:element name="getDescription">
4601             <xsd:annotation>
4602                 <xsd:documentation xml:lang="en">get the description of the ReadPoint.</xsd:documentation>
4603             </xsd:annotation>
4604             <xsd:complexType/>
4605         </xsd:element>
4606         <xsd:element name="setDescription">
4607             <xsd:annotation>
4608                 <xsd:documentation xml:lang="en">set the ReadPoint description.</xsd:documentation>
4609             </xsd:annotation>
4610             <xsd:complexType>
4611                 <xsd:sequence>
4612                     <xsd:element name="description" type="xsd:string">
4613                         <xsd:annotation>
4614                             <xsd:documentation xml:lang="en"> description to set.</xsd:documentation>
4615                         </xsd:annotation>
4616                     </xsd:element>
4617                 </xsd:sequence>
4618             </xsd:complexType>
4619         </xsd:element>
4620         <xsd:element name="getAdminStatus">
4621             <xsd:annotation>
4622                 <xsd:documentation xml:lang="en">get the current ReadPoint administrative status.</xsd:documentation>
4623             </xsd:annotation>
4624             <xsd:complexType/>
4625         </xsd:element>
4626         <xsd:element name="setAdminStatus">
4627             <xsd:annotation>
4628                 <xsd:documentation xml:lang="en">set the current ReadPoint administrative status.</xsd:documentation>
4629             </xsd:annotation>
4630             <xsd:complexType>
4631                 <xsd:sequence>
4632                     <xsd:element name="status" type="rm:AdministrativeStatusParamType">
4633                         <xsd:annotation>
4634                             <xsd:documentation xml:lang="en"> administrative status to set. </xsd:documentation>
4635                         </xsd:annotation>
4636                     </xsd:element>
4637                 </xsd:sequence>

```

```

4638     </xsd:complexType>
4639 </xsd:element>
4640 <xsd:element name="getOperStatus">
4641   <xsd:annotation>
4642     <xsd:documentation xml:lang="en">get the ReadPoint's current operational status.</xsd:documentation>
4643   </xsd:annotation>
4644   <xsd:complexType/>
4645 </xsd:element>
4646 <xsd:element name="getOperStatusAlarmControl">
4647   <xsd:annotation>
4648     <xsd:documentation xml:lang="en">get the ReadPoint's operational status alarm control.
4649 </xsd:documentation>
4650   </xsd:annotation>
4651   <xsd:complexType/>
4652 </xsd:element>
4653 <xsd:any namespace="##any" processContents="lax">
4654   <xsd:annotation>
4655     <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4656   </xsd:annotation>
4657 </xsd:any>
4658 </xsd:choice>
4659 </xsd:complexType>
4660 <xsd:complexType name="SourceCommand">
4661   <xsd:choice>
4662     <xsd:annotation>
4663       <xsd:documentation xml:lang="en">Source object commands. </xsd:documentation>
4664     </xsd:annotation>
4665     <xsd:element name="getUnknownToGlimpsedCount">
4666       <xsd:annotation>
4667         <xsd:documentation xml:lang="en">query the reader for the number of times a transition from state
4668 Unknown to state Glimpsed have been detected for the particular source.</xsd:documentation>
4669       </xsd:annotation>
4670       <xsd:complexType/>
4671     </xsd:element>
4672     <xsd:element name="getGlimpsedToUnknownCount">
4673       <xsd:annotation>
4674         <xsd:documentation xml:lang="en">query the reader for the number of times a transition from state
4675 Glimpsed to state Unknown have been detected for the particular source.</xsd:documentation>
4676       </xsd:annotation>
4677       <xsd:complexType/>
4678     </xsd:element>
4679     <xsd:element name="getGlimpsedToObservedCount">
4680       <xsd:annotation>
4681         <xsd:documentation xml:lang="en">query the reader for the number of times a transition from state
4682 Glimpsed to state Observed have been detected for the particular source.</xsd:documentation>
4683       </xsd:annotation>
4684       <xsd:complexType/>
4685     </xsd:element>
4686     <xsd:element name="getObservedToLostCount">
4687       <xsd:annotation>
4688         <xsd:documentation xml:lang="en">query the reader for the number of times a transition from state
4689 Observed to state Lost have been detected for the particular source.</xsd:documentation>
4690       </xsd:annotation>
4691       <xsd:complexType/>
4692     </xsd:element>
4693     <xsd:element name="getLostToGlimpsedCount">

```

```

4694     <xsd:annotation>
4695         <xsd:documentation xml:lang="en">query the reader for the number of times a transition from state Lost to
4696 state Glimpsed have been detected for the particular source.</xsd:documentation>
4697     </xsd:annotation>
4698     <xsd:complexType/>
4699 </xsd:element>
4700 <xsd:element name="getLostToUnknownCount">
4701     <xsd:annotation>
4702         <xsd:documentation xml:lang="en">query the reader for the number of times a transition from state Lost to
4703 state Unknown have been detected for the particular source.</xsd:documentation>
4704     </xsd:annotation>
4705     <xsd:complexType/>
4706 </xsd:element>
4707 <xsd:element name="getOperStatus">
4708     <xsd:annotation>
4709         <xsd:documentation xml:lang="en">query the reader for the operational status of this particular
4710 Source.</xsd:documentation>
4711     </xsd:annotation>
4712     <xsd:complexType/>
4713 </xsd:element>
4714 <xsd:element name="getAdminStatus">
4715     <xsd:annotation>
4716         <xsd:documentation xml:lang="en">query the reader for the administrative status of this particular
4717 Source.</xsd:documentation>
4718     </xsd:annotation>
4719     <xsd:complexType/>
4720 </xsd:element>
4721 <xsd:element name="setAdminStatus">
4722     <xsd:annotation>
4723         <xsd:documentation xml:lang="en">set the administrative status of this particular
4724 Source.</xsd:documentation>
4725     </xsd:annotation>
4726     <xsd:complexType>
4727         <xsd:sequence>
4728             <xsd:element name="administrativeStatus" type="rm:AdministrativeStatusParamType">
4729                 <xsd:annotation>
4730                     <xsd:documentation xml:lang="en">administrative status to set.</xsd:documentation>
4731                 </xsd:annotation>
4732             </xsd:element>
4733         </xsd:sequence>
4734     </xsd:complexType>
4735 </xsd:element>
4736 <xsd:element name="getOperStatusAlarmControl">
4737     <xsd:annotation>
4738         <xsd:documentation xml:lang="en">get this particular source's operational status alarm control.
4739 </xsd:documentation>
4740     </xsd:annotation>
4741     <xsd:complexType/>
4742 </xsd:element>
4743 <xsd:any namespace="##any" processContents="lax">
4744     <xsd:annotation>
4745         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4746     </xsd:annotation>
4747 </xsd:any>
4748 </xsd:choice>
4749 </xsd:complexType>

```

```

4750 <xsd:complexType name="TriggerCommand">
4751   <xsd:choice>
4752     <xsd:annotation>
4753       <xsd:documentation xml:lang="en"> Trigger object commands. </xsd:documentation>
4754     </xsd:annotation>
4755     <xsd:element name="getFireCount">
4756       <xsd:annotation>
4757         <xsd:documentation xml:lang="en">query the reader for the number of times a particular trigger has
4758 fired.</xsd:documentation>
4759       </xsd:annotation>
4760     </xsd:complexType/>
4761   </xsd:element>
4762   <xsd:any namespace="##any" processContents="lax">
4763     <xsd:annotation>
4764       <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4765     </xsd:annotation>
4766   </xsd:any>
4767 </xsd:choice>
4768 </xsd:complexType>
4769 <xsd:complexType name="TTOperationalStatusAlarmControlCommand">
4770   <xsd:choice>
4771     <xsd:annotation>
4772       <xsd:documentation xml:lang="en"> Transition triggered alarm control object commands.
4773 </xsd:documentation>
4774     </xsd:annotation>
4775     <xsd:element name="getTriggerFromState">
4776       <xsd:annotation>
4777         <xsd:documentation xml:lang="en">query the reader for the current value of the
4778 TTOperationalStatusAlarmControlâ€™s TriggerFromState attribute.</xsd:documentation>
4779       </xsd:annotation>
4780     </xsd:complexType/>
4781   </xsd:element>
4782   <xsd:element name="setTriggerFromState">
4783     <xsd:annotation>
4784       <xsd:documentation xml:lang="en">set the current value of the TTOperationalStatusAlarmControlâ€™s
4785 TriggerFromState attribute.</xsd:documentation>
4786     </xsd:annotation>
4787     <xsd:complexType>
4788       <xsd:sequence>
4789         <xsd:element name="triggerFromState" type="rm:OperationalStatusParamType">
4790           <xsd:annotation>
4791             <xsd:documentation xml:lang="en">trigger from state to set</xsd:documentation>
4792           </xsd:annotation>
4793         </xsd:element>
4794       </xsd:sequence>
4795     </xsd:complexType>
4796   </xsd:element>
4797   <xsd:element name="getTriggerToState">
4798     <xsd:annotation>
4799       <xsd:documentation xml:lang="en">query the reader for the current value of the
4800 TTOperationalStatusAlarmControlâ€™s TriggerToState attribute.</xsd:documentation>
4801     </xsd:annotation>
4802     <xsd:complexType/>
4803   </xsd:element>
4804   <xsd:element name="setTriggerToState">
4805     <xsd:annotation>

```

```

4806      <xsd:documentation xml:lang="en">set the current value of the TOperatonaStatusAlarmControlâ™s
4807 TriggerToState attribute.</xsd:documentation>
4808      </xsd:annotation>
4809      <xsd:complexType>
4810      <xsd:sequence>
4811      <xsd:element name="triggerToState" type="rm:OperationalStatusParamType">
4812      <xsd:annotation>
4813      <xsd:documentation xml:lang="en">trigger to state to set</xsd:documentation>
4814      </xsd:annotation>
4815      </xsd:element>
4816      </xsd:sequence>
4817      </xsd:complexType>
4818      </xsd:element>
4819      <xsd:any namespace="##any" processContents="lax">
4820      <xsd:annotation>
4821      <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4822      </xsd:annotation>
4823      </xsd:any>
4824      </xsd:choice>
4825      </xsd:complexType>
4826      </xsd:schema>
4827

```

## 4828 10.3.2 Reply XML Message Encoding (Reader-To-Host)

```

4829 <?xml version="1.0" encoding="UTF-8"?>
4830 <xsd:schema targetNamespace="urn:epcglobal:rm:xsd:1" elementFormDefault="unqualified"
4831 attributeFormDefault="unqualified" version="1.0" xmlns:rm="urn:epcglobal:rm:xsd:1"
4832 xmlns:epcglobal="urn:epcglobal:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4833 <xsd:annotation>
4834 <xsd:documentation xml:lang="en">
4835 <epcglobal:copyright>Copyright&#169;2005-2006 Epcglobal Inc., All Rights Reserved.</epcglobal:copyright>
4836 <epcglobal:disclaimer>EPCglobal Inc., its members, officers, directors, employees, or
4837 agents shall not be liable for any injury, loss, damages, financial or otherwise,
4838 arising from, related to, or caused by the use of this document. The use of said
4839 document shall constitute your express consent to the foregoing exculpation.</epcglobal:disclaimer>
4840 <epcglobal:specification>Reader Management (RM) version 1.0</epcglobal:specification>
4841 </xsd:documentation>
4842 </xsd:annotation>
4843 <xsd:include schemaLocation="RmCommon.xsd"/>
4844 <xsd:import namespace="urn:epcglobal:xsd:1" schemaLocation="./EpcGlobal.xsd"/>
4845 <!-- Reader Management Reply -->
4846 <xsd:element name="reply">
4847 <xsd:annotation>
4848 <xsd:documentation xml:lang="en"> This element defines a reader management reply. Just like commands,
4849 replies are grouped by the object they belong to. To specify the object a reply relates to, the name of the object is
4850 used. All the replies contain a single return value inside the reply. return value is enclosed by the element
4851 "returnValue". In case of a single return value it is directly added to this element. In case of an array of return
4852 values they are enclosed within element
4853 "list" with each value in turn enclosed in element "value". Each reply is wrapped in
4854 its object type. </xsd:documentation>
4855 </xsd:annotation>
4856 <xsd:complexType>
4857 <xsd:complexContent>
4858 <xsd:extension base="epcglobal:Document">

```

```

4859     <xsd:sequence>
4860         <xsd:element name="id" type="xsd:string">
4861             <xsd:annotation>
4862                 <xsd:documentation xml:lang="en">The id of the command for which this is the
4863 reply.</xsd:documentation>
4864             </xsd:annotation>
4865         </xsd:element>
4866         <xsd:element name="resultCode" type="xsd:int">
4867             <xsd:annotation>
4868                 <xsd:documentation xml:lang="en"> the error code for the command - 0 for
4869 success.</xsd:documentation>
4870             </xsd:annotation>
4871         </xsd:element>
4872     <xsd:choice>
4873         <!-- Error information -->
4874         <xsd:element name="error" type="rm:ErrorType">
4875             <xsd:annotation>
4876                 <xsd:documentation xml:lang="en">Information about the error.</xsd:documentation>
4877             </xsd:annotation>
4878         </xsd:element>
4879         <!-- Reader Management (Target) Object Types -->
4880         <!-- elements inside are ordered in alphabetical order -->
4881         <xsd:element name="alarmChannel" type="rm:AlarmChannelReply">
4882             <xsd:annotation>
4883                 <xsd:documentation xml:lang="en">An Alarm channel</xsd:documentation>
4884             </xsd:annotation>
4885         </xsd:element>
4886         <xsd:element name="alarmControl" type="rm:AlarmControlReply">
4887             <xsd:annotation>
4888                 <xsd:documentation xml:lang="en">AlarmControl is the base class for all of classes
4889 within the RMP object model responsible for controlling the generation of alarm messages within an EPCglobal
4890 compliant Reader.</xsd:documentation>
4891             </xsd:annotation>
4892         </xsd:element>
4893         <xsd:element name="antennaReadPoint" type="rm:AntennaReadPointReply">
4894             <xsd:annotation>
4895                 <xsd:documentation xml:lang="en">Extension of Read point object for
4896 antenna.</xsd:documentation>
4897             </xsd:annotation>
4898         </xsd:element>
4899         <xsd:element name="edgeTriggeredAlarmControl" type="rm:EdgeTriggeredAlarmControlReply">
4900             <xsd:annotation>
4901                 <xsd:documentation xml:lang="en">This class extends AlarmControl to control alarms
4902 generated when a monitored, integer-valued, model element first crosses a threshold value (the
4903 AlarmThreshold).</xsd:documentation>
4904             </xsd:annotation>
4905         </xsd:element>
4906         <xsd:element name="iOPort" type="rm:IOPortReply">
4907             <xsd:annotation>
4908                 <xsd:documentation xml:lang="en">The hardware element that provides external input
4909 and output lines to connect to other components outside the reader device.</xsd:documentation>
4910             </xsd:annotation>
4911         </xsd:element>
4912         <xsd:element name="notificationChannel" type="rm:NotificationChannelReply">
4913             <xsd:annotation>
4914                 <xsd:documentation xml:lang="en">The notification channel

```

```

4915         carries messages issued asynchronously by the Reader to the Host.</xsd:documentation>
4916     </xsd:annotation>
4917 </xsd:element>
4918 <xsd:element name="readerDevice" type="rm:ReaderDeviceReply">
4919     <xsd:annotation>
4920         <xsd:documentation xml:lang="en">A Reader</xsd:documentation>
4921     </xsd:annotation>
4922 </xsd:element>
4923 <xsd:element name="readPoint" type="rm:ReadPointReply">
4924     <xsd:annotation>
4925         <xsd:documentation xml:lang="en">A ReadPoint</xsd:documentation>
4926     </xsd:annotation>
4927 </xsd:element>
4928 <xsd:element name="source" type="rm:SourceReply">
4929     <xsd:annotation>
4930         <xsd:documentation xml:lang="en">A Read source</xsd:documentation>
4931     </xsd:annotation>
4932 </xsd:element>
4933 <xsd:element name="trigger" type="rm:TriggerReply">
4934     <xsd:annotation>
4935         <xsd:documentation xml:lang="en">A Read/Notify Trigger</xsd:documentation>
4936     </xsd:annotation>
4937 </xsd:element>
4938 <xsd:element name="tOperationalStatusAlarmControl"
4939 type="rm:TOperationalStatusAlarmControlReply">
4940     <xsd:annotation>
4941         <xsd:documentation xml:lang="en">This class extends AlarmControl to control alarms
4942 generated when a monitored model element of type OperationalStatus transitions to a new
4943 value.</xsd:documentation>
4944     </xsd:annotation>
4945 </xsd:element>
4946 <xsd:any namespace="##any" processContents="lax">
4947     <xsd:annotation>
4948         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4949     </xsd:annotation>
4950 </xsd:any>
4951 </xsd:choice>
4952 </xsd:sequence>
4953 </xsd:extension>
4954 </xsd:complexContent>
4955 </xsd:complexType>
4956 </xsd:element>
4957 <xsd:complexType name="ErrorType">
4958     <xsd:annotation>
4959         <xsd:documentation xml:lang="en">Information about the error.</xsd:documentation>
4960     </xsd:annotation>
4961     <xsd:sequence>
4962         <xsd:element name="name" type="xsd:string" minOccurs="0">
4963             <xsd:annotation>
4964                 <xsd:documentation xml:lang="en">Name of the error.</xsd:documentation>
4965             </xsd:annotation>
4966         </xsd:element>
4967         <xsd:element name="cause" type="xsd:string" minOccurs="0">
4968             <xsd:annotation>
4969                 <xsd:documentation xml:lang="en">Cause of the error.</xsd:documentation>
4970             </xsd:annotation>

```

```

4971     </xsd:element>
4972     <xsd:element name="description" type="xsd:string" minOccurs="0">
4973         <xsd:annotation>
4974             <xsd:documentation xml:lang="en">Description of the error.</xsd:documentation>
4975         </xsd:annotation>
4976     </xsd:element>
4977     <xsd:any namespace="##any" processContents="lax">
4978         <xsd:annotation>
4979             <xsd:documentation>For standard and vendor extensions</xsd:documentation>
4980         </xsd:annotation>
4981     </xsd:any>
4982 </xsd:sequence>
4983 </xsd:complexType>
4984 <!-- Reader Management (Target) Object Types -->
4985 <!-- types are ordered in alphabetical order -->
4986 <!-- inside each type replies are listed in the order in which it is defined in the RM specification. -->
4987 <xsd:complexType name="AlarmChannelReply">
4988     <xsd:choice>
4989         <xsd:annotation>
4990             <xsd:documentation xml:lang="en"> Alarm channel object replies.</xsd:documentation>
4991         </xsd:annotation>
4992         <xsd:element name="create">
4993             <xsd:annotation>
4994                 <xsd:documentation xml:lang="en">reply for create an Alarm channel.</xsd:documentation>
4995             </xsd:annotation>
4996             <xsd:complexType>
4997                 <xsd:sequence>
4998                     <xsd:element name="returnValue" type="rm:AlarmChannelParamType">
4999                         <xsd:annotation>
5000                             <xsd:documentation xml:lang="en"> created alarm channel.</xsd:documentation>
5001                         </xsd:annotation>
5002                     </xsd:element>
5003                 </xsd:sequence>
5004             </xsd:complexType>
5005         </xsd:element>
5006         <xsd:element name="getName">
5007             <xsd:annotation>
5008                 <xsd:documentation xml:lang="en"> reply for get Name.</xsd:documentation>
5009             </xsd:annotation>
5010             <xsd:complexType>
5011                 <xsd:sequence>
5012                     <xsd:element name="returnValue" type="xsd:string">
5013                         <xsd:annotation>
5014                             <xsd:documentation xml:lang="en">This AlarmChannel's name</xsd:documentation>
5015                         </xsd:annotation>
5016                     </xsd:element>
5017                 </xsd:sequence>
5018             </xsd:complexType>
5019         </xsd:element>
5020         <xsd:element name="getAddress">
5021             <xsd:annotation>
5022                 <xsd:documentation xml:lang="en"> reply for get Address.</xsd:documentation>
5023             </xsd:annotation>
5024             <xsd:complexType>
5025                 <xsd:sequence>
5026                     <xsd:element name="returnValue" type="rm:AddressParamType">

```



```

5027         <xsd:annotation>
5028             <xsd:documentation xml:lang="en">reporting address for this AlarmChannel.
5029 </xsd:documentation>
5030         </xsd:annotation>
5031     </xsd:element>
5032 </xsd:sequence>
5033 </xsd:complexType>
5034 </xsd:element>
5035 <xsd:element name="setAddress">
5036     <xsd:annotation>
5037         <xsd:documentation xml:lang="en">reply for set Address</xsd:documentation>
5038     </xsd:annotation>
5039     <xsd:complexType/>
5040 </xsd:element>
5041 <xsd:any namespace="##any" processContents="lax">
5042     <xsd:annotation>
5043         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
5044     </xsd:annotation>
5045 </xsd:any>
5046 </xsd:choice>
5047 </xsd:complexType>
5048 <xsd:complexType name="AlarmControlReply">
5049     <xsd:choice>
5050         <xsd:annotation>
5051             <xsd:documentation xml:lang="en"> Alarm control object replies. </xsd:documentation>
5052         </xsd:annotation>
5053         <xsd:element name="getName">
5054             <xsd:annotation>
5055                 <xsd:documentation xml:lang="en"> reply for get Name.</xsd:documentation>
5056             </xsd:annotation>
5057             <xsd:complexType>
5058                 <xsd:sequence>
5059                     <xsd:element name="returnValue" type="xsd:string">
5060                         <xsd:annotation>
5061                             <xsd:documentation xml:lang="en">This AlarmControl's name</xsd:documentation>
5062                         </xsd:annotation>
5063                     </xsd:element>
5064                 </xsd:sequence>
5065             </xsd:complexType>
5066         </xsd:element>
5067         <xsd:element name="getEnabled">
5068             <xsd:annotation>
5069                 <xsd:documentation xml:lang="en">reply for get enabled.</xsd:documentation>
5070             </xsd:annotation>
5071             <xsd:complexType>
5072                 <xsd:sequence>
5073                     <xsd:element name="returnValue" type="xsd:boolean">
5074                         <xsd:annotation>
5075                             <xsd:documentation xml:lang="en">Whether alarm control is enabled.</xsd:documentation>
5076                         </xsd:annotation>
5077                     </xsd:element>
5078                 </xsd:sequence>
5079             </xsd:complexType>
5080         </xsd:element>
5081         <xsd:element name="setEnabled">
5082             <xsd:annotation>

```

```

5083     <xsd:documentation xml:lang="en">reply for set enabled.</xsd:documentation>
5084     </xsd:annotation>
5085     <xsd:complexType/>
5086 </xsd:element>
5087 <xsd:element name="getLevel">
5088     <xsd:annotation>
5089     <xsd:documentation xml:lang="en">reply for get Level.</xsd:documentation>
5090     </xsd:annotation>
5091     <xsd:complexType>
5092     <xsd:sequence>
5093     <xsd:element name="returnValue" type="rm:AlarmLevelParamType">
5094     <xsd:annotation>
5095     <xsd:documentation xml:lang="en">current value of the AlarmControl's
5096 Level.</xsd:documentation>
5097     </xsd:annotation>
5098     </xsd:element>
5099     </xsd:sequence>
5100 </xsd:complexType>
5101 </xsd:element>
5102 <xsd:element name="setLevel">
5103     <xsd:annotation>
5104     <xsd:documentation xml:lang="en">reply for set Level.</xsd:documentation>
5105     </xsd:annotation>
5106     <xsd:complexType/>
5107 </xsd:element>
5108 <xsd:element name="getSuppressInterval">
5109     <xsd:annotation>
5110     <xsd:documentation xml:lang="en">reply for get SuppressInterval.</xsd:documentation>
5111     </xsd:annotation>
5112     <xsd:complexType>
5113     <xsd:sequence>
5114     <xsd:element name="returnValue" type="xsd:int">
5115     <xsd:annotation>
5116     <xsd:documentation xml:lang="en">current value of the AlarmControl's SuppressInterval in
5117 seconds.</xsd:documentation>
5118     </xsd:annotation>
5119     </xsd:element>
5120     </xsd:sequence>
5121 </xsd:complexType>
5122 </xsd:element>
5123 <xsd:element name="setSuppressInterval">
5124     <xsd:annotation>
5125     <xsd:documentation xml:lang="en">reply for set SuppressInterval.</xsd:documentation>
5126     </xsd:annotation>
5127     <xsd:complexType/>
5128 </xsd:element>
5129 <xsd:any namespace="##any" processContents="lax">
5130     <xsd:annotation>
5131     <xsd:documentation>For standard and vendor extensions</xsd:documentation>
5132     </xsd:annotation>
5133 </xsd:any>
5134 </xsd:choice>
5135 </xsd:complexType>
5136 <xsd:complexType name="AntennaReadPointReply">
5137 <xsd:choice>
5138 <xsd:annotation>

```

```

5139     <xsd:documentation xml:lang="en"> Antenna read point object replies </xsd:documentation>
5140 </xsd:annotation>
5141 <xsd:element name="getIdentificationCount">
5142   <xsd:annotation>
5143     <xsd:documentation xml:lang="en">reply for get identification count.</xsd:documentation>
5144   </xsd:annotation>
5145   <xsd:complexType>
5146     <xsd:sequence>
5147       <xsd:element name="returnValue" type="xsd:int">
5148         <xsd:annotation>
5149           <xsd:documentation xml:lang="en">The count of the successful tag identifiers read at this
5150 AntennaReadPoint.</xsd:documentation>
5151         </xsd:annotation>
5152       </xsd:element>
5153     </xsd:sequence>
5154   </xsd:complexType>
5155 </xsd:element>
5156 <xsd:element name="getFailedIdentificationCount">
5157   <xsd:annotation>
5158     <xsd:documentation xml:lang="en">reply for get failed identification count.</xsd:documentation>
5159   </xsd:annotation>
5160   <xsd:complexType>
5161     <xsd:sequence>
5162       <xsd:element name="returnValue" type="xsd:int">
5163         <xsd:annotation>
5164           <xsd:documentation xml:lang="en">The count of the failed attempts at reading the identifier for a
5165 tag at this antenna ReadPoint.</xsd:documentation>
5166         </xsd:annotation>
5167       </xsd:element>
5168     </xsd:sequence>
5169   </xsd:complexType>
5170 </xsd:element>
5171 <xsd:element name="getMemReadCount">
5172   <xsd:annotation>
5173     <xsd:documentation xml:lang="en">reply for get mem read count.</xsd:documentation>
5174   </xsd:annotation>
5175   <xsd:complexType>
5176     <xsd:sequence>
5177       <xsd:element name="returnValue" type="xsd:int">
5178         <xsd:annotation>
5179           <xsd:documentation xml:lang="en">The count of the successful tag memory reads at this antenna
5180 ReadPoint.</xsd:documentation>
5181         </xsd:annotation>
5182       </xsd:element>
5183     </xsd:sequence>
5184   </xsd:complexType>
5185 </xsd:element>
5186 <xsd:element name="getFailedMemReadCount">
5187   <xsd:annotation>
5188     <xsd:documentation xml:lang="en">reply for get failed mem read count.</xsd:documentation>
5189   </xsd:annotation>
5190   <xsd:complexType>
5191     <xsd:sequence>
5192       <xsd:element name="returnValue" type="xsd:int">
5193         <xsd:annotation>

```

```

5194         <xsd:documentation xml:lang="en">The count of the failed tag memory reads at this antenna
5195 ReadPoint.</xsd:documentation>
5196         </xsd:annotation>
5197     </xsd:element>
5198 </xsd:sequence>
5199 </xsd:complexType>
5200 </xsd:element>
5201 <xsd:element name="getFailedMemReadAlarmControl">
5202     <xsd:annotation>
5203         <xsd:documentation xml:lang="en">reply for get failed mem read alarm control.</xsd:documentation>
5204     </xsd:annotation>
5205     <xsd:complexType>
5206         <xsd:sequence>
5207             <xsd:element name="returnValue" type="rm:AlarmControlParamType">
5208                 <xsd:annotation>
5209                     <xsd:documentation xml:lang="en">An alarm control for monitoring tag memory read
5210 failures.</xsd:documentation>
5211                 </xsd:annotation>
5212             </xsd:element>
5213         </xsd:sequence>
5214     </xsd:complexType>
5215 </xsd:element>
5216 <xsd:element name="getWriteCount">
5217     <xsd:annotation>
5218         <xsd:documentation xml:lang="en">reply for get write count.</xsd:documentation>
5219     </xsd:annotation>
5220     <xsd:complexType>
5221         <xsd:sequence>
5222             <xsd:element name="returnValue" type="xsd:int">
5223                 <xsd:annotation>
5224                     <xsd:documentation xml:lang="en">The count of the successful writes at this
5225 AntennaReadPoint.</xsd:documentation>
5226                 </xsd:annotation>
5227             </xsd:element>
5228         </xsd:sequence>
5229     </xsd:complexType>
5230 </xsd:element>
5231 <xsd:element name="getFailedWriteCount">
5232     <xsd:annotation>
5233         <xsd:documentation xml:lang="en">reply for get failed write count.</xsd:documentation>
5234     </xsd:annotation>
5235     <xsd:complexType>
5236         <xsd:sequence>
5237             <xsd:element name="returnValue" type="xsd:int">
5238                 <xsd:annotation>
5239                     <xsd:documentation xml:lang="en">The count of the failed writes at this
5240 AntennaReadPoint.</xsd:documentation>
5241                 </xsd:annotation>
5242             </xsd:element>
5243         </xsd:sequence>
5244     </xsd:complexType>
5245 </xsd:element>
5246 <xsd:element name="getFailedWriteAlarmControl">
5247     <xsd:annotation>
5248         <xsd:documentation xml:lang="en">reply for get failed write alarm control.</xsd:documentation>
5249     </xsd:annotation>

```

```

5250     <xsd:complexType>
5251         <xsd:sequence>
5252             <xsd:element name="returnValue" type="rm:AlarmControlParamType">
5253                 <xsd:annotation>
5254                     <xsd:documentation xml:lang="en">An alarm control for monitoring the number of failed
5255 writes.</xsd:documentation>
5256                 </xsd:annotation>
5257             </xsd:element>
5258         </xsd:sequence>
5259     </xsd:complexType>
5260 </xsd:element>
5261 <xsd:element name="getKillCount">
5262     <xsd:annotation>
5263         <xsd:documentation xml:lang="en">reply for get kill count.</xsd:documentation>
5264     </xsd:annotation>
5265     <xsd:complexType>
5266         <xsd:sequence>
5267             <xsd:element name="returnValue" type="xsd:int">
5268                 <xsd:annotation>
5269                     <xsd:documentation xml:lang="en">The count of the successful tag kills at this antenna
5270 ReadPoint.</xsd:documentation>
5271                 </xsd:annotation>
5272             </xsd:element>
5273         </xsd:sequence>
5274     </xsd:complexType>
5275 </xsd:element>
5276 <xsd:element name="getFailedKillCount">
5277     <xsd:annotation>
5278         <xsd:documentation xml:lang="en">reply for get failed kill count.</xsd:documentation>
5279     </xsd:annotation>
5280     <xsd:complexType>
5281         <xsd:sequence>
5282             <xsd:element name="returnValue" type="xsd:int">
5283                 <xsd:annotation>
5284                     <xsd:documentation xml:lang="en">The count of the failed tag kills at this antenna
5285 readpoint.</xsd:documentation>
5286                 </xsd:annotation>
5287             </xsd:element>
5288         </xsd:sequence>
5289     </xsd:complexType>
5290 </xsd:element>
5291 <xsd:element name="getFailedKillAlarmControl">
5292     <xsd:annotation>
5293         <xsd:documentation xml:lang="en">reply for get failed kill alarm control.</xsd:documentation>
5294     </xsd:annotation>
5295     <xsd:complexType>
5296         <xsd:sequence>
5297             <xsd:element name="returnValue" type="rm:AlarmControlParamType">
5298                 <xsd:annotation>
5299                     <xsd:documentation xml:lang="en">An alarm control for monitoring tag kill
5300 failures.</xsd:documentation>
5301                 </xsd:annotation>
5302             </xsd:element>
5303         </xsd:sequence>
5304     </xsd:complexType>
5305 </xsd:element>

```

```

5306 <xsd:element name="getEraseCount">
5307   <xsd:annotation>
5308     <xsd:documentation xml:lang="en">reply for get erase count.</xsd:documentation>
5309   </xsd:annotation>
5310   <xsd:complexType>
5311     <xsd:sequence>
5312       <xsd:element name="returnValue" type="xsd:int">
5313         <xsd:annotation>
5314           <xsd:documentation xml:lang="en">The count of the successful tag erasures at this antenna
5315 ReadPoint.</xsd:documentation>
5316         </xsd:annotation>
5317       </xsd:element>
5318     </xsd:sequence>
5319   </xsd:complexType>
5320 </xsd:element>
5321 <xsd:element name="getFailedEraseCount">
5322   <xsd:annotation>
5323     <xsd:documentation xml:lang="en">reply for get failed erase count.</xsd:documentation>
5324   </xsd:annotation>
5325   <xsd:complexType>
5326     <xsd:sequence>
5327       <xsd:element name="returnValue" type="xsd:int">
5328         <xsd:annotation>
5329           <xsd:documentation xml:lang="en">The count of the failed tag erasures at this antenna
5330 readpoint.</xsd:documentation>
5331         </xsd:annotation>
5332       </xsd:element>
5333     </xsd:sequence>
5334   </xsd:complexType>
5335 </xsd:element>
5336 <xsd:element name="getFailedEraseAlarmControl">
5337   <xsd:annotation>
5338     <xsd:documentation xml:lang="en">reply for get failed erase alarm control.</xsd:documentation>
5339   </xsd:annotation>
5340   <xsd:complexType>
5341     <xsd:sequence>
5342       <xsd:element name="returnValue" type="rm:AlarmControlParamType">
5343         <xsd:annotation>
5344           <xsd:documentation xml:lang="en">An alarm control for monitoring tag erasure
5345 failures.</xsd:documentation>
5346         </xsd:annotation>
5347       </xsd:element>
5348     </xsd:sequence>
5349   </xsd:complexType>
5350 </xsd:element>
5351 <xsd:element name="getLockCount">
5352   <xsd:annotation>
5353     <xsd:documentation xml:lang="en">reply for get lock count.</xsd:documentation>
5354   </xsd:annotation>
5355   <xsd:complexType>
5356     <xsd:sequence>
5357       <xsd:element name="returnValue" type="xsd:int">
5358         <xsd:annotation>
5359           <xsd:documentation xml:lang="en">The count of the successful tag locks at this antenna
5360 ReadPoint.</xsd:documentation>
5361         </xsd:annotation>

```

```

5362     </xsd:element>
5363     </xsd:sequence>
5364     </xsd:complexType>
5365 </xsd:element>
5366 <xsd:element name="getFailedLockCount">
5367     <xsd:annotation>
5368         <xsd:documentation xml:lang="en">reply for get failed lock count.</xsd:documentation>
5369     </xsd:annotation>
5370     <xsd:complexType>
5371         <xsd:sequence>
5372             <xsd:element name="returnValue" type="xsd:int">
5373                 <xsd:annotation>
5374                     <xsd:documentation xml:lang="en">The count of the failed tag locks at this antenna
5375 readpoint.</xsd:documentation>
5376                 </xsd:annotation>
5377             </xsd:element>
5378         </xsd:sequence>
5379     </xsd:complexType>
5380 </xsd:element>
5381 <xsd:element name="getFailedLockAlarmControl">
5382     <xsd:annotation>
5383         <xsd:documentation xml:lang="en">reply for get failed lock alarm control.</xsd:documentation>
5384     </xsd:annotation>
5385     <xsd:complexType>
5386         <xsd:sequence>
5387             <xsd:element name="returnValue" type="rm:AlarmControlParamType">
5388                 <xsd:annotation>
5389                     <xsd:documentation xml:lang="en">An alarm control for monitoring tag lock
5390 failures.</xsd:documentation>
5391                 </xsd:annotation>
5392             </xsd:element>
5393         </xsd:sequence>
5394     </xsd:complexType>
5395 </xsd:element>
5396 <xsd:element name="getTimeEnergized">
5397     <xsd:annotation>
5398         <xsd:documentation xml:lang="en">reply for get time energized.</xsd:documentation>
5399     </xsd:annotation>
5400     <xsd:complexType>
5401         <xsd:sequence>
5402             <xsd:element name="returnValue" type="xsd:int">
5403                 <xsd:annotation>
5404                     <xsd:documentation xml:lang="en">The number of milliseconds the AntennaReadPoint has been
5405 energized attempting communication with tags.</xsd:documentation>
5406                 </xsd:annotation>
5407             </xsd:element>
5408         </xsd:sequence>
5409     </xsd:complexType>
5410 </xsd:element>
5411 <xsd:element name="getPowerLevel">
5412     <xsd:annotation>
5413         <xsd:documentation xml:lang="en">reply for get power level.</xsd:documentation>
5414     </xsd:annotation>
5415     <xsd:complexType>
5416         <xsd:sequence>
5417             <xsd:element name="returnValue" type="xsd:int">

```

```

5418         <xsd:annotation>
5419             <xsd:documentation xml:lang="en">current transmit power level of the antenna
5420 ReadPoint.</xsd:documentation>
5421         </xsd:annotation>
5422     </xsd:element>
5423 </xsd:sequence>
5424 </xsd:complexType>
5425 </xsd:element>
5426 <xsd:element name="getNoiseLevel">
5427     <xsd:annotation>
5428         <xsd:documentation xml:lang="en">reply for get noise level.</xsd:documentation>
5429     </xsd:annotation>
5430     <xsd:complexType>
5431         <xsd:sequence>
5432             <xsd:element name="returnValue" type="xsd:int">
5433                 <xsd:annotation>
5434                     <xsd:documentation xml:lang="en">current noise level observed at the antenna
5435 readpoint.</xsd:documentation>
5436                 </xsd:annotation>
5437             </xsd:element>
5438         </xsd:sequence>
5439     </xsd:complexType>
5440 </xsd:element>
5441 <xsd:any namespace="##any" processContents="lax">
5442     <xsd:annotation>
5443         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
5444     </xsd:annotation>
5445 </xsd:any>
5446 </xsd:choice>
5447 </xsd:complexType>
5448 <xsd:complexType name="EdgeTriggeredAlarmControlReply">
5449     <xsd:choice>
5450         <xsd:annotation>
5451             <xsd:documentation xml:lang="en"> Edge triggered alarm control object replies. </xsd:documentation>
5452         </xsd:annotation>
5453         <xsd:element name="getAlarmThreshold">
5454             <xsd:annotation>
5455                 <xsd:documentation xml:lang="en">reply for get Alarm threshold.</xsd:documentation>
5456             </xsd:annotation>
5457             <xsd:complexType>
5458                 <xsd:sequence>
5459                     <xsd:element name="returnValue" type="xsd:int">
5460                         <xsd:annotation>
5461                             <xsd:documentation xml:lang="en">current value of the EdgeTriggeredAlarmControl's
5462 AlarmThreshold .</xsd:documentation>
5463                         </xsd:annotation>
5464                     </xsd:element>
5465                 </xsd:sequence>
5466             </xsd:complexType>
5467         </xsd:element>
5468         <xsd:element name="setAlarmThreshold">
5469             <xsd:annotation>
5470                 <xsd:documentation xml:lang="en">reply for set alarm threshold.</xsd:documentation>
5471             </xsd:annotation>
5472             <xsd:complexType/>
5473         </xsd:element>

```



```

5474 <xsd:element name="getRearmThreshold">
5475   <xsd:annotation>
5476     <xsd:documentation xml:lang="en">reply for get RearmThreshold.</xsd:documentation>
5477   </xsd:annotation>
5478   <xsd:complexType>
5479     <xsd:sequence>
5480       <xsd:element name="returnValue" type="xsd:int">
5481         <xsd:annotation>
5482           <xsd:documentation xml:lang="en">current value of the EdgeTriggeredAlarmControl's
5483 RearmThreshold.</xsd:documentation>
5484         </xsd:annotation>
5485       </xsd:element>
5486     </xsd:sequence>
5487   </xsd:complexType>
5488 </xsd:element>
5489 <xsd:element name="setRearmThreshold">
5490   <xsd:annotation>
5491     <xsd:documentation xml:lang="en">reply for set RearmThreshold.</xsd:documentation>
5492   </xsd:annotation>
5493   <xsd:complexType/>
5494 </xsd:element>
5495 <xsd:element name="getDirection">
5496   <xsd:annotation>
5497     <xsd:documentation xml:lang="en">reply for get direction.</xsd:documentation>
5498   </xsd:annotation>
5499   <xsd:complexType>
5500     <xsd:sequence>
5501       <xsd:element name="returnValue" type="rm:EdgeTriggeredAlarmDirectionParamType">
5502         <xsd:annotation>
5503           <xsd:documentation xml:lang="en">current value of the EdgeTriggeredAlarmControl's
5504 Direction.</xsd:documentation>
5505         </xsd:annotation>
5506       </xsd:element>
5507     </xsd:sequence>
5508   </xsd:complexType>
5509 </xsd:element>
5510 <xsd:element name="setDirection">
5511   <xsd:annotation>
5512     <xsd:documentation xml:lang="en">reply for set direction.</xsd:documentation>
5513   </xsd:annotation>
5514   <xsd:complexType/>
5515 </xsd:element>
5516 <xsd:element name="getStatus">
5517   <xsd:annotation>
5518     <xsd:documentation xml:lang="en">reply for get status.</xsd:documentation>
5519   </xsd:annotation>
5520   <xsd:complexType>
5521     <xsd:sequence>
5522       <xsd:element name="returnValue" type="rm:EdgeTriggeredAlarmStatusParamType">
5523         <xsd:annotation>
5524           <xsd:documentation xml:lang="en">current value of the EdgeTriggeredAlarmControl's
5525 Status.</xsd:documentation>
5526         </xsd:annotation>
5527       </xsd:element>
5528     </xsd:sequence>
5529   </xsd:complexType>

```

```

5530     </xsd:element>
5531     <xsd:any namespace="##any" processContents="lax">
5532         <xsd:annotation>
5533             <xsd:documentation>For standard and vendor extensions</xsd:documentation>
5534         </xsd:annotation>
5535     </xsd:any>
5536 </xsd:choice>
5537 </xsd:complexType>
5538 <xsd:complexType name="IOPortReply">
5539     <xsd:choice>
5540         <xsd:annotation>
5541             <xsd:documentation xml:lang="en">IO port object replies. </xsd:documentation>
5542         </xsd:annotation>
5543         <xsd:element name="getName">
5544             <xsd:annotation>
5545                 <xsd:documentation xml:lang="en"> reply for get Name.</xsd:documentation>
5546             </xsd:annotation>
5547             <xsd:complexType>
5548                 <xsd:sequence>
5549                     <xsd:element name="returnValue" type="xsd:string">
5550                         <xsd:annotation>
5551                             <xsd:documentation xml:lang="en">This IO port's name</xsd:documentation>
5552                         </xsd:annotation>
5553                     </xsd:element>
5554                 </xsd:sequence>
5555             </xsd:complexType>
5556         </xsd:element>
5557         <xsd:element name="getDescription">
5558             <xsd:annotation>
5559                 <xsd:documentation xml:lang="en">reply for get description.</xsd:documentation>
5560             </xsd:annotation>
5561             <xsd:complexType>
5562                 <xsd:sequence>
5563                     <xsd:element name="returnValue" type="xsd:string">
5564                         <xsd:annotation>
5565                             <xsd:documentation xml:lang="en">a textual description of the IO-port.</xsd:documentation>
5566                         </xsd:annotation>
5567                     </xsd:element>
5568                 </xsd:sequence>
5569             </xsd:complexType>
5570         </xsd:element>
5571         <xsd:element name="setDescription">
5572             <xsd:annotation>
5573                 <xsd:documentation xml:lang="en">reply for set description.</xsd:documentation>
5574             </xsd:annotation>
5575             <xsd:complexType/>
5576         </xsd:element>
5577         <xsd:element name="getOperStatus">
5578             <xsd:annotation>
5579                 <xsd:documentation xml:lang="en">reply for get operational status.</xsd:documentation>
5580             </xsd:annotation>
5581             <xsd:complexType>
5582                 <xsd:sequence>
5583                     <xsd:element name="returnValue" type="rm:OperationalStatusParamType">
5584                         <xsd:annotation>
5585                             <xsd:documentation xml:lang="en">the operational status of the IO-port.</xsd:documentation>

```

```

5586         </xsd:annotation>
5587     </xsd:element>
5588 </xsd:sequence>
5589 </xsd:complexType>
5590 </xsd:element>
5591 <xsd:element name="getAdminStatus">
5592     <xsd:annotation>
5593         <xsd:documentation xml:lang="en">reply for get administrative status.</xsd:documentation>
5594     </xsd:annotation>
5595 <xsd:complexType>
5596     <xsd:sequence>
5597         <xsd:element name="returnValue" type="rm:AdministrativeStatusParamType">
5598             <xsd:annotation>
5599                 <xsd:documentation xml:lang="en">the administrative status of the IO-port.</xsd:documentation>
5600             </xsd:annotation>
5601         </xsd:element>
5602     </xsd:sequence>
5603 </xsd:complexType>
5604 </xsd:element>
5605 <xsd:element name="setAdminStatus">
5606     <xsd:annotation>
5607         <xsd:documentation xml:lang="en">reply for set admin status.</xsd:documentation>
5608     </xsd:annotation>
5609 <xsd:complexType/>
5610 </xsd:element>
5611 <xsd:element name="getOperStatusAlarmControl">
5612     <xsd:annotation>
5613         <xsd:documentation xml:lang="en">reply for get operational status alarm control.</xsd:documentation>
5614     </xsd:annotation>
5615 <xsd:complexType>
5616     <xsd:sequence>
5617         <xsd:element name="returnValue" type="rm:TTOperationalStatusAlarmControlParamType">
5618             <xsd:annotation>
5619                 <xsd:documentation xml:lang="en">An alarm control for monitoring the operational status of the
5620 IOPort.</xsd:documentation>
5621             </xsd:annotation>
5622         </xsd:element>
5623     </xsd:sequence>
5624 </xsd:complexType>
5625 </xsd:element>
5626 <xsd:any namespace="##any" processContents="lax">
5627     <xsd:annotation>
5628         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
5629     </xsd:annotation>
5630 </xsd:any>
5631 </xsd:choice>
5632 </xsd:complexType>
5633 <xsd:complexType name="NotificationChannelReply">
5634     <xsd:choice>
5635         <xsd:annotation>
5636             <xsd:documentation xml:lang="en">Notification channel object replies</xsd:documentation>
5637         </xsd:annotation>
5638         <xsd:element name="getLastNotificationAttempt">
5639             <xsd:annotation>
5640                 <xsd:documentation xml:lang="en">reply for get last notification attempt.</xsd:documentation>
5641             </xsd:annotation>

```

```

5642     <xsd:complexType>
5643     <xsd:sequence>
5644         <xsd:element name="returnValue" type="xsd:int">
5645             <xsd:annotation>
5646                 <xsd:documentation xml:lang="en">the timestamp (TimeTicks) when the last attempt was made to
5647 send a notification to the given address.</xsd:documentation>
5648             </xsd:annotation>
5649         </xsd:element>
5650     </xsd:sequence>
5651 </xsd:complexType>
5652 </xsd:element>
5653 <xsd:element name="getLastSuccessfulNotification">
5654     <xsd:annotation>
5655         <xsd:documentation xml:lang="en">reply for get last successful notification.</xsd:documentation>
5656     </xsd:annotation>
5657     <xsd:complexType>
5658         <xsd:sequence>
5659             <xsd:element name="returnValue" type="xsd:int">
5660                 <xsd:annotation>
5661                     <xsd:documentation xml:lang="en">the timestamp (TimeTicks) when the last successful
5662 notification was send to the given address.</xsd:documentation>
5663                 </xsd:annotation>
5664             </xsd:element>
5665         </xsd:sequence>
5666     </xsd:complexType>
5667 </xsd:element>
5668 <xsd:element name="getOperStatus">
5669     <xsd:annotation>
5670         <xsd:documentation xml:lang="en">reply for get operational status.</xsd:documentation>
5671     </xsd:annotation>
5672     <xsd:complexType>
5673         <xsd:sequence>
5674             <xsd:element name="returnValue" type="rm:OperationalStatusParamType">
5675                 <xsd:annotation>
5676                     <xsd:documentation xml:lang="en">the operational status of the notification
5677 channel.</xsd:documentation>
5678                 </xsd:annotation>
5679             </xsd:element>
5680         </xsd:sequence>
5681     </xsd:complexType>
5682 </xsd:element>
5683 <xsd:element name="setAdminStatus">
5684     <xsd:annotation>
5685         <xsd:documentation xml:lang="en">reply for set administrative status.</xsd:documentation>
5686     </xsd:annotation>
5687     <xsd:complexType/>
5688 </xsd:element>
5689 <xsd:element name="getAdminStatus">
5690     <xsd:annotation>
5691         <xsd:documentation xml:lang="en">reply for get administrative status.</xsd:documentation>
5692     </xsd:annotation>
5693     <xsd:complexType>
5694         <xsd:sequence>
5695             <xsd:element name="returnValue" type="rm:AdministrativeStatusParamType">
5696                 <xsd:annotation>

```

```

5697         <xsd:documentation xml:lang="en">the administrative status of the notification
5698 channel.</xsd:documentation>
5699         </xsd:annotation>
5700     </xsd:element>
5701 </xsd:sequence>
5702 </xsd:complexType>
5703 </xsd:element>
5704 <xsd:element name="getOperStatusAlarmControl">
5705     <xsd:annotation>
5706         <xsd:documentation xml:lang="en">reply for get operational status alarm control.</xsd:documentation>
5707     </xsd:annotation>
5708     <xsd:complexType>
5709         <xsd:sequence>
5710             <xsd:element name="returnValue" type="rm:TTOperationalStatusAlarmControlParamType">
5711                 <xsd:annotation>
5712                     <xsd:documentation xml:lang="en">An alarm control for monitoring the operational status of the
5713 notification channel.</xsd:documentation>
5714                 </xsd:annotation>
5715             </xsd:element>
5716         </xsd:sequence>
5717     </xsd:complexType>
5718 </xsd:element>
5719 <xsd:any namespace="##any" processContents="lax">
5720     <xsd:annotation>
5721         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
5722     </xsd:annotation>
5723 </xsd:any>
5724 </xsd:choice>
5725 </xsd:complexType>
5726 <xsd:complexType name="ReaderDeviceReply">
5727     <xsd:annotation>
5728         <xsd:documentation xml:lang="en">Reader Device object replies. </xsd:documentation>
5729     </xsd:annotation>
5730     <xsd:choice>
5731         <xsd:element name="getDescription">
5732             <xsd:annotation>
5733                 <xsd:documentation xml:lang="en">reply for get description.</xsd:documentation>
5734             </xsd:annotation>
5735             <xsd:complexType>
5736                 <xsd:sequence>
5737                     <xsd:element name="returnValue" type="xsd:string">
5738                         <xsd:annotation>
5739                             <xsd:documentation xml:lang="en">user defined description of the reader.</xsd:documentation>
5740                         </xsd:annotation>
5741                     </xsd:element>
5742                 </xsd:sequence>
5743             </xsd:complexType>
5744         </xsd:element>
5745         <xsd:element name="setDescription">
5746             <xsd:annotation>
5747                 <xsd:documentation xml:lang="en">reply for set description.</xsd:documentation>
5748             </xsd:annotation>
5749             <xsd:complexType/>
5750         </xsd:element>
5751         <xsd:element name="getLocationDescription">
5752             <xsd:annotation>

```

```

5753     <xsd:documentation xml:lang="en">reply for get location description.</xsd:documentation>
5754 </xsd:annotation>
5755 <xsd:complexType>
5756   <xsd:sequence>
5757     <xsd:element name="returnValue" type="xsd:string">
5758       <xsd:annotation>
5759         <xsd:documentation xml:lang="en">user defined location description of the reader
5760 device.</xsd:documentation>
5761       </xsd:annotation>
5762     </xsd:element>
5763   </xsd:sequence>
5764 </xsd:complexType>
5765 </xsd:element>
5766 <xsd:element name="setLocationDescription">
5767   <xsd:annotation>
5768     <xsd:documentation xml:lang="en">reply for set location description.</xsd:documentation>
5769   </xsd:annotation>
5770 </xsd:complexType/>
5771 </xsd:element>
5772 <xsd:element name="getContact">
5773   <xsd:annotation>
5774     <xsd:documentation xml:lang="en">reply for get contact.</xsd:documentation>
5775   </xsd:annotation>
5776 <xsd:complexType>
5777   <xsd:sequence>
5778     <xsd:element name="returnValue" type="xsd:string">
5779       <xsd:annotation>
5780         <xsd:documentation xml:lang="en">user defined contact description.</xsd:documentation>
5781       </xsd:annotation>
5782     </xsd:element>
5783   </xsd:sequence>
5784 </xsd:complexType>
5785 </xsd:element>
5786 <xsd:element name="setContact">
5787   <xsd:annotation>
5788     <xsd:documentation xml:lang="en">reply for set contact.</xsd:documentation>
5789   </xsd:annotation>
5790 </xsd:complexType/>
5791 </xsd:element>
5792 <xsd:element name="getSerialNumber">
5793   <xsd:annotation>
5794     <xsd:documentation xml:lang="en">reply for get serial number.</xsd:documentation>
5795   </xsd:annotation>
5796 <xsd:complexType>
5797   <xsd:sequence>
5798     <xsd:element name="returnValue" type="xsd:string">
5799       <xsd:annotation>
5800         <xsd:documentation xml:lang="en">the serial number of the reader device.</xsd:documentation>
5801       </xsd:annotation>
5802     </xsd:element>
5803   </xsd:sequence>
5804 </xsd:complexType>
5805 </xsd:element>
5806 <xsd:element name="getOperStatus">
5807   <xsd:annotation>
5808     <xsd:documentation xml:lang="en">reply for get operational status.</xsd:documentation>

```

```

5809     </xsd:annotation>
5810     <xsd:complexType>
5811       <xsd:sequence>
5812         <xsd:element name="returnValue" type="rm:OperationalStatusParamType">
5813           <xsd:annotation>
5814             <xsd:documentation xml:lang="en">the operational status of the reader.</xsd:documentation>
5815           </xsd:annotation>
5816         </xsd:element>
5817       </xsd:sequence>
5818     </xsd:complexType>
5819   </xsd:element>
5820   <xsd:element name="getOperStatusAlarmControl">
5821     <xsd:annotation>
5822       <xsd:documentation xml:lang="en">reply for get operational status alarm control.</xsd:documentation>
5823     </xsd:annotation>
5824     <xsd:complexType>
5825       <xsd:sequence>
5826         <xsd:element name="returnValue" type="rm:TTOperationalStatusAlarmControlParamType">
5827           <xsd:annotation>
5828             <xsd:documentation xml:lang="en">An alarm control for monitoring the operational status of the
5829 reader.</xsd:documentation>
5830           </xsd:annotation>
5831         </xsd:element>
5832       </xsd:sequence>
5833     </xsd:complexType>
5834   </xsd:element>
5835   <xsd:element name="getFreeMemory">
5836     <xsd:annotation>
5837       <xsd:documentation xml:lang="en">reply for get free memory.</xsd:documentation>
5838     </xsd:annotation>
5839     <xsd:complexType>
5840       <xsd:sequence>
5841         <xsd:element name="returnValue" type="xsd:int">
5842           <xsd:annotation>
5843             <xsd:documentation xml:lang="en">available free memory.</xsd:documentation>
5844           </xsd:annotation>
5845         </xsd:element>
5846       </xsd:sequence>
5847     </xsd:complexType>
5848   </xsd:element>
5849   <xsd:element name="getFreeMemoryAlarmControl">
5850     <xsd:annotation>
5851       <xsd:documentation xml:lang="en">reply for get free memory alarm control.</xsd:documentation>
5852     </xsd:annotation>
5853     <xsd:complexType>
5854       <xsd:sequence>
5855         <xsd:element name="returnValue" type="rm:EdgeTriggeredAlarmControlParamType">
5856           <xsd:annotation>
5857             <xsd:documentation xml:lang="en">FreeMemoryAlarmControl.</xsd:documentation>
5858           </xsd:annotation>
5859         </xsd:element>
5860       </xsd:sequence>
5861     </xsd:complexType>
5862   </xsd:element>
5863   <xsd:element name="getNTPServers">
5864     <xsd:annotation>

```

```

5865     <xsd:documentation xml:lang="en">reply for get NTP servers.</xsd:documentation>
5866 </xsd:annotation>
5867 <xsd:complexType>
5868   <xsd:sequence>
5869     <xsd:element name="returnValue">
5870       <xsd:complexType>
5871         <xsd:sequence>
5872           <xsd:element name="list">
5873             <xsd:complexType>
5874               <xsd:sequence>
5875                 <xsd:element name="value" type="xsd:string"
5876 minOccurs="0" maxOccurs="unbounded">
5877                   <xsd:annotation>
5878                     <xsd:documentation xml:lang="en">
5879 NTP servers used by the device to synchronize its current UTC clock (TimeUTC). </xsd:documentation>
5880                   </xsd:annotation>
5881                 </xsd:element>
5882               </xsd:sequence>
5883             </xsd:complexType>
5884           </xsd:element>
5885         </xsd:sequence>
5886       </xsd:complexType>
5887     </xsd:element>
5888   </xsd:sequence>
5889 </xsd:complexType>
5890 </xsd:element>
5891 <xsd:element name="getDHCPserver">
5892   <xsd:annotation>
5893     <xsd:documentation xml:lang="en">reply for get DHCP server.</xsd:documentation>
5894   </xsd:annotation>
5895   <xsd:complexType>
5896     <xsd:sequence>
5897       <xsd:element name="returnValue" type="xsd:string">
5898         <xsd:annotation>
5899           <xsd:documentation xml:lang="en">the DHCP server currently used by the device for DHCP
5900 requests.</xsd:documentation>
5901         </xsd:annotation>
5902       </xsd:element>
5903     </xsd:sequence>
5904   </xsd:complexType>
5905 </xsd:element>
5906 <xsd:element name="getIOPort">
5907   <xsd:annotation>
5908     <xsd:documentation xml:lang="en">reply for get IO port.</xsd:documentation>
5909   </xsd:annotation>
5910   <xsd:complexType>
5911     <xsd:sequence>
5912       <xsd:element name="returnValue" type="rm:IOPortParamType">
5913         <xsd:annotation>
5914           <xsd:documentation xml:lang="en">the named IO port object.</xsd:documentation>
5915         </xsd:annotation>
5916       </xsd:element>
5917     </xsd:sequence>
5918   </xsd:complexType>
5919 </xsd:element>
5920 <xsd:element name="getAllIOPorts">

```



```

5921 <xsd:annotation>
5922   <xsd:documentation xml:lang="en">reply for get all IO ports.</xsd:documentation>
5923 </xsd:annotation>
5924 <xsd:complexType>
5925   <xsd:sequence>
5926     <xsd:element name="returnValue">
5927       <xsd:complexType>
5928         <xsd:sequence>
5929           <xsd:element name="list">
5930             <xsd:complexType>
5931               <xsd:sequence>
5932                 <xsd:element name="value" type="rm:IOPortParamType"
5933 minOccurs="0" maxOccurs="unbounded">
5934                   <xsd:annotation>
5935                     <xsd:documentation xml:lang="en">
5936 IOPort objects.</xsd:documentation>
5937                   </xsd:annotation>
5938                 </xsd:element>
5939               </xsd:sequence>
5940             </xsd:complexType>
5941           </xsd:element>
5942         </xsd:sequence>
5943       </xsd:complexType>
5944     </xsd:element>
5945   </xsd:sequence>
5946 </xsd:complexType>
5947 </xsd:element>
5948 <xsd:element name="resetStatistics">
5949   <xsd:annotation>
5950     <xsd:documentation xml:lang="en">reply for reset statistics.</xsd:documentation>
5951   </xsd:annotation>
5952   <xsd:complexType/>
5953 </xsd:element>
5954 <xsd:element name="removeAlarmChannels">
5955   <xsd:annotation>
5956     <xsd:documentation xml:lang="en">reply for remove alarm channels.</xsd:documentation>
5957   </xsd:annotation>
5958   <xsd:complexType/>
5959 </xsd:element>
5960 <xsd:element name="removeAllAlarmChannels">
5961   <xsd:annotation>
5962     <xsd:documentation xml:lang="en">reply for remove all alarm channels.</xsd:documentation>
5963   </xsd:annotation>
5964   <xsd:complexType/>
5965 </xsd:element>
5966 <xsd:element name="getAlarmChannel">
5967   <xsd:annotation>
5968     <xsd:documentation xml:lang="en">reply for get alarm channel.</xsd:documentation>
5969   </xsd:annotation>
5970   <xsd:complexType>
5971     <xsd:sequence>
5972       <xsd:element name="returnValue" type="rm:AlarmChannelParamType">
5973         <xsd:annotation>
5974           <xsd:documentation xml:lang="en">the named AlarmChannel object.</xsd:documentation>
5975         </xsd:annotation>
5976       </xsd:element>

```

```

5977     </xsd:sequence>
5978     </xsd:complexType>
5979 </xsd:element>
5980 <xsd:element name="getAllAlarmChannels">
5981     <xsd:annotation>
5982         <xsd:documentation xml:lang="en">reply for get all alarm channels.</xsd:documentation>
5983     </xsd:annotation>
5984     <xsd:complexType>
5985         <xsd:sequence>
5986             <xsd:element name="returnValue">
5987                 <xsd:complexType>
5988                     <xsd:sequence>
5989                         <xsd:element name="list">
5990                             <xsd:complexType>
5991                                 <xsd:sequence>
5992                                     <xsd:element name="value"
5993 type="rm:AlarmChannelParamType" minOccurs="0" maxOccurs="unbounded">
5994                                         <xsd:annotation>
5995                                             <xsd:documentation xml:lang="en">
5996 Alarm channels. </xsd:documentation>
5997                                         </xsd:annotation>
5998                                     </xsd:element>
5999                                 </xsd:sequence>
6000                             </xsd:complexType>
6001                         </xsd:element>
6002                     </xsd:sequence>
6003                 </xsd:complexType>
6004             </xsd:element>
6005         </xsd:sequence>
6006     </xsd:complexType>
6007 </xsd:element>
6008 <xsd:any namespace="##any" processContents="lax">
6009     <xsd:annotation>
6010         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
6011     </xsd:annotation>
6012 </xsd:any>
6013 </xsd:choice>
6014 </xsd:complexType>
6015 <xsd:complexType name="ReadPointReply">
6016     <xsd:annotation>
6017         <xsd:documentation xml:lang="en"> Read point object replies.</xsd:documentation>
6018     </xsd:annotation>
6019     <xsd:choice>
6020         <xsd:element name="getClassName">
6021             <xsd:annotation>
6022                 <xsd:documentation xml:lang="en">reply for get class name.</xsd:documentation>
6023             </xsd:annotation>
6024             <xsd:complexType>
6025                 <xsd:sequence>
6026                     <xsd:element name="returnValue" type="xsd:string">
6027                         <xsd:annotation>
6028                             <xsd:documentation xml:lang="en">The class name for the ReadPoint. The only supported return
6029 value is "AntennaReadPoint". </xsd:documentation>
6030                         </xsd:annotation>
6031                     </xsd:element>
6032                 </xsd:sequence>

```

```

6033     </xsd:complexType>
6034 </xsd:element>
6035 <xsd:element name="getDescription">
6036   <xsd:annotation>
6037     <xsd:documentation xml:lang="en">reply for get description.</xsd:documentation>
6038   </xsd:annotation>
6039   <xsd:complexType>
6040     <xsd:sequence>
6041       <xsd:element name="returnValue" type="xsd:string">
6042         <xsd:annotation>
6043           <xsd:documentation xml:lang="en">The readpoint descriptive name.</xsd:documentation>
6044         </xsd:annotation>
6045       </xsd:element>
6046     </xsd:sequence>
6047   </xsd:complexType>
6048 </xsd:element>
6049 <xsd:element name="setDescription">
6050   <xsd:annotation>
6051     <xsd:documentation xml:lang="en">reply for set description.</xsd:documentation>
6052   </xsd:annotation>
6053   <xsd:complexType/>
6054 </xsd:element>
6055 <xsd:element name="getAdminStatus">
6056   <xsd:annotation>
6057     <xsd:documentation xml:lang="en">reply for get administrative status.</xsd:documentation>
6058   </xsd:annotation>
6059   <xsd:complexType>
6060     <xsd:sequence>
6061       <xsd:element name="returnValue" type="rm:AdministrativeStatusParamType">
6062         <xsd:annotation>
6063           <xsd:documentation xml:lang="en">the administrative status of the read
6064 point.</xsd:documentation>
6065         </xsd:annotation>
6066       </xsd:element>
6067     </xsd:sequence>
6068   </xsd:complexType>
6069 </xsd:element>
6070 <xsd:element name="setAdminStatus">
6071   <xsd:annotation>
6072     <xsd:documentation xml:lang="en">reply for set administrative status.</xsd:documentation>
6073   </xsd:annotation>
6074   <xsd:complexType/>
6075 </xsd:element>
6076 <xsd:element name="getOperStatus">
6077   <xsd:annotation>
6078     <xsd:documentation xml:lang="en">reply for get operational status.</xsd:documentation>
6079   </xsd:annotation>
6080   <xsd:complexType>
6081     <xsd:sequence>
6082       <xsd:element name="returnValue" type="rm:OperationalStatusParamType">
6083         <xsd:annotation>
6084           <xsd:documentation xml:lang="en">the operational status of the read point.</xsd:documentation>
6085         </xsd:annotation>
6086       </xsd:element>
6087     </xsd:sequence>
6088   </xsd:complexType>

```

```

6089 </xsd:element>
6090 <xsd:element name="getOperStatusAlarmControl">
6091 <xsd:annotation>
6092 <xsd:documentation xml:lang="en">reply for get operational status alarm control.</xsd:documentation>
6093 </xsd:annotation>
6094 <xsd:complexType>
6095 <xsd:sequence>
6096 <xsd:element name="returnValue" type="rm:TTOperationalStatusAlarmControlParamType">
6097 <xsd:annotation>
6098 <xsd:documentation xml:lang="en">An alarm control for monitoring the operational status of the
6099 read point.</xsd:documentation>
6100 </xsd:annotation>
6101 </xsd:element>
6102 </xsd:sequence>
6103 </xsd:complexType>
6104 </xsd:element>
6105 <xsd:any namespace="##any" processContents="lax">
6106 <xsd:annotation>
6107 <xsd:documentation>For standard and vendor extensions</xsd:documentation>
6108 </xsd:annotation>
6109 </xsd:any>
6110 </xsd:choice>
6111 </xsd:complexType>
6112 <xsd:complexType name="SourceReply">
6113 <xsd:choice>
6114 <xsd:annotation>
6115 <xsd:documentation xml:lang="en">Source object replies. </xsd:documentation>
6116 </xsd:annotation>
6117 <xsd:element name="getUnknownToGlimpsedCount">
6118 <xsd:annotation>
6119 <xsd:documentation xml:lang="en">reply for get unknown to glimpsed count.</xsd:documentation>
6120 </xsd:annotation>
6121 <xsd:complexType>
6122 <xsd:sequence>
6123 <xsd:element name="returnValue" type="xsd:int">
6124 <xsd:annotation>
6125 <xsd:documentation xml:lang="en">the number of times a transition from state Unknown to state
6126 Glimpsed have been detected for the particular source.</xsd:documentation>
6127 </xsd:annotation>
6128 </xsd:element>
6129 </xsd:sequence>
6130 </xsd:complexType>
6131 </xsd:element>
6132 <xsd:element name="getGlimpsedToUnknownCount">
6133 <xsd:annotation>
6134 <xsd:documentation xml:lang="en">reply for get glimpsed to unknown count.</xsd:documentation>
6135 </xsd:annotation>
6136 <xsd:complexType>
6137 <xsd:sequence>
6138 <xsd:element name="returnValue" type="xsd:int">
6139 <xsd:annotation>
6140 <xsd:documentation xml:lang="en">the number of times a transition from state Glimpsed to state
6141 Unknown have been detected for the particular source.</xsd:documentation>
6142 </xsd:annotation>
6143 </xsd:element>
6144 </xsd:sequence>

```

```

6145     </xsd:complexType>
6146 </xsd:element>
6147 <xsd:element name="getGlimpsedToObservedCount">
6148   <xsd:annotation>
6149     <xsd:documentation xml:lang="en">reply for get Glimpsed to Observed count.</xsd:documentation>
6150   </xsd:annotation>
6151   <xsd:complexType>
6152     <xsd:sequence>
6153       <xsd:element name="returnValue" type="xsd:int">
6154         <xsd:annotation>
6155           <xsd:documentation xml:lang="en">the number of times a transition from state Glimpsed to state
6156 Observed have been detected for the particular source.</xsd:documentation>
6157         </xsd:annotation>
6158       </xsd:element>
6159     </xsd:sequence>
6160   </xsd:complexType>
6161 </xsd:element>
6162 <xsd:element name="getObservedToLostCount">
6163   <xsd:annotation>
6164     <xsd:documentation xml:lang="en">reply for get Observed to Lost count.</xsd:documentation>
6165   </xsd:annotation>
6166   <xsd:complexType>
6167     <xsd:sequence>
6168       <xsd:element name="returnValue" type="xsd:int">
6169         <xsd:annotation>
6170           <xsd:documentation xml:lang="en">the number of times a transition from state Observed to state
6171 Lost have been detected for the particular source.</xsd:documentation>
6172         </xsd:annotation>
6173       </xsd:element>
6174     </xsd:sequence>
6175   </xsd:complexType>
6176 </xsd:element>
6177 <xsd:element name="getLostToGlimpsedCount">
6178   <xsd:annotation>
6179     <xsd:documentation xml:lang="en">reply for get Lost to Glimpsed count.</xsd:documentation>
6180   </xsd:annotation>
6181   <xsd:complexType>
6182     <xsd:sequence>
6183       <xsd:element name="returnValue" type="xsd:int">
6184         <xsd:annotation>
6185           <xsd:documentation xml:lang="en">the number of times a transition from state Lost to state
6186 Observed have been detected for the particular source.</xsd:documentation>
6187         </xsd:annotation>
6188       </xsd:element>
6189     </xsd:sequence>
6190   </xsd:complexType>
6191 </xsd:element>
6192 <xsd:element name="getLostToUnknownCount">
6193   <xsd:annotation>
6194     <xsd:documentation xml:lang="en">reply for get Lost to Unknown count.</xsd:documentation>
6195   </xsd:annotation>
6196   <xsd:complexType>
6197     <xsd:sequence>
6198       <xsd:element name="returnValue" type="xsd:int">
6199         <xsd:annotation>

```

```

6200         <xsd:documentation xml:lang="en">the number of times a transition from state Lost to state
6201 Unknown have been detected for the particular source.</xsd:documentation>
6202     </xsd:annotation>
6203 </xsd:element>
6204 </xsd:sequence>
6205 </xsd:complexType>
6206 </xsd:element>
6207 <xsd:element name="getOperStatus">
6208     <xsd:annotation>
6209         <xsd:documentation xml:lang="en">reply for get operational status.</xsd:documentation>
6210     </xsd:annotation>
6211     <xsd:complexType>
6212         <xsd:sequence>
6213             <xsd:element name="returnValue" type="rm:OperationalStatusParamType">
6214                 <xsd:annotation>
6215                     <xsd:documentation xml:lang="en">the operational status of the source.</xsd:documentation>
6216                 </xsd:annotation>
6217             </xsd:element>
6218         </xsd:sequence>
6219     </xsd:complexType>
6220 </xsd:element>
6221 <xsd:element name="getAdminStatus">
6222     <xsd:annotation>
6223         <xsd:documentation xml:lang="en">reply for get administrative status.</xsd:documentation>
6224     </xsd:annotation>
6225     <xsd:complexType>
6226         <xsd:sequence>
6227             <xsd:element name="returnValue" type="rm:AdministrativeStatusParamType">
6228                 <xsd:annotation>
6229                     <xsd:documentation xml:lang="en">the administrative status of the source.</xsd:documentation>
6230                 </xsd:annotation>
6231             </xsd:element>
6232         </xsd:sequence>
6233     </xsd:complexType>
6234 </xsd:element>
6235 <xsd:element name="setAdminStatus">
6236     <xsd:annotation>
6237         <xsd:documentation xml:lang="en">reply for administrative status.</xsd:documentation>
6238     </xsd:annotation>
6239     <xsd:complexType/>
6240 </xsd:element>
6241 <xsd:element name="getOperStatusAlarmControl">
6242     <xsd:annotation>
6243         <xsd:documentation xml:lang="en">reply for get operational status alarm control.</xsd:documentation>
6244     </xsd:annotation>
6245     <xsd:complexType>
6246         <xsd:sequence>
6247             <xsd:element name="returnValue" type="rm:TTOperationalStatusAlarmControlParamType">
6248                 <xsd:annotation>
6249                     <xsd:documentation xml:lang="en">An alarm control for monitoring the operational status of
6250 source.</xsd:documentation>
6251                 </xsd:annotation>
6252             </xsd:element>
6253         </xsd:sequence>
6254     </xsd:complexType>
6255 </xsd:element>

```

```

6256     <xsd:any namespace="##any" processContents="lax">
6257         <xsd:annotation>
6258             <xsd:documentation>For standard and vendor extensions</xsd:documentation>
6259         </xsd:annotation>
6260     </xsd:any>
6261 </xsd:choice>
6262 </xsd:complexType>
6263 <xsd:complexType name="TriggerReply">
6264     <xsd:choice>
6265         <xsd:annotation>
6266             <xsd:documentation xml:lang="en"> Trigger object replies. </xsd:documentation>
6267         </xsd:annotation>
6268         <xsd:element name="getFireCount">
6269             <xsd:annotation>
6270                 <xsd:documentation xml:lang="en">reply for get fire count.</xsd:documentation>
6271             </xsd:annotation>
6272             <xsd:complexType>
6273                 <xsd:sequence>
6274                     <xsd:element name="returnValue" type="xsd:int">
6275                         <xsd:annotation>
6276                             <xsd:documentation xml:lang="en">the number of times a particular trigger has
6277 fired.</xsd:documentation>
6278                         </xsd:annotation>
6279                     </xsd:element>
6280                 </xsd:sequence>
6281             </xsd:complexType>
6282         </xsd:element>
6283     <xsd:any namespace="##any" processContents="lax">
6284         <xsd:annotation>
6285             <xsd:documentation>For standard and vendor extensions</xsd:documentation>
6286         </xsd:annotation>
6287     </xsd:any>
6288 </xsd:choice>
6289 </xsd:complexType>
6290 <xsd:complexType name="TTOperationalStatusAlarmControlReply">
6291     <xsd:choice>
6292         <xsd:annotation>
6293             <xsd:documentation xml:lang="en"> Transition triggered alarm control object replies. </xsd:documentation>
6294         </xsd:annotation>
6295         <xsd:element name="getTriggerFromState">
6296             <xsd:annotation>
6297                 <xsd:documentation xml:lang="en">reply for get trigger from state.</xsd:documentation>
6298             </xsd:annotation>
6299             <xsd:complexType>
6300                 <xsd:sequence>
6301                     <xsd:element name="returnValue" type="rm:OperationalStatusParamType">
6302                         <xsd:annotation>
6303                             <xsd:documentation xml:lang="en">current value of the TTOperationalStatusAlarmControl's
6304 TriggerFromState.</xsd:documentation>
6305                         </xsd:annotation>
6306                     </xsd:element>
6307                 </xsd:sequence>
6308             </xsd:complexType>
6309         </xsd:element>
6310         <xsd:element name="setTriggerFromState">
6311             <xsd:annotation>

```

```

6312     <xsd:documentation xml:lang="en">reply for set trigger from state.</xsd:documentation>
6313     </xsd:annotation>
6314     <xsd:complexType/>
6315 </xsd:element>
6316 <xsd:element name="getTriggerToState">
6317     <xsd:annotation>
6318         <xsd:documentation xml:lang="en">reply for get trigger to state.</xsd:documentation>
6319     </xsd:annotation>
6320     <xsd:complexType>
6321         <xsd:sequence>
6322             <xsd:element name="returnValue" type="rm:OperationalStatusParamType">
6323                 <xsd:annotation>
6324                     <xsd:documentation xml:lang="en">current value of the TTOperationalStatusAlarmControl's
6325 TriggerToState.</xsd:documentation>
6326                 </xsd:annotation>
6327             </xsd:element>
6328         </xsd:sequence>
6329     </xsd:complexType>
6330 </xsd:element>
6331 <xsd:element name="setTriggerToState">
6332     <xsd:annotation>
6333         <xsd:documentation xml:lang="en">reply for set trigger to state.</xsd:documentation>
6334     </xsd:annotation>
6335     <xsd:complexType/>
6336 </xsd:element>
6337 <xsd:any namespace="##any" processContents="lax">
6338     <xsd:annotation>
6339         <xsd:documentation>For standard and vendor extensions</xsd:documentation>
6340     </xsd:annotation>
6341 </xsd:any>
6342 </xsd:choice>
6343 </xsd:complexType>
6344 </xsd:schema>
6345

```

### 6346 **10.3.3 Alarm Notification XML Message Encoding (Reader-** 6347 **To-Host)**

```

6348 <?xml version="1.0" encoding="UTF-8"?>
6349 <xsd:schema targetNamespace="urn:epcglobal:rm:xsd:1" elementFormDefault="unqualified"
6350 attributeFormDefault="unqualified" version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6351 xmlns:epcglobal="urn:epcglobal:xsd:1" xmlns:rm="urn:epcglobal:rm:xsd:1">
6352     <xsd:annotation>
6353         <xsd:documentation xml:lang="en">
6354             <epcglobal:copyright>Copyright&#169;2005-2006 Epcglobal Inc., All Rights Reserved.</epcglobal:copyright>
6355             <epcglobal:disclaimer>EPCglobal Inc., its members, officers, directors, employees, or
6356             agents shall not be liable for any injury, loss, damages, financial or otherwise,
6357             arising from, related to, or caused by the use of this document. The use of said
6358             document shall constitute your express consent to the foregoing exculpation.</epcglobal:disclaimer>
6359             <epcglobal:specification>Reader Management (RM) version 1.0</epcglobal:specification>
6360         </xsd:documentation>
6361     </xsd:annotation>
6362     <xsd:include schemaLocation="RmCommon.xsd"/>
6363     <xsd:import namespace="urn:epcglobal:xsd:1" schemaLocation="./EpcGlobal.xsd"/>
6364     <!-- Reader Management Alarm -->

```



```

6365 <xsd:element name="alarm" type="rm:AlarmType">
6366   <xsd:annotation>
6367     <xsd:documentation xml:lang="en"> This element defines a reader management alarm.</xsd:documentation>
6368   </xsd:annotation>
6369 </xsd:element>
6370 <xsd:complexType name="ReaderType">
6371   <xsd:annotation>
6372     <xsd:documentation xml:lang="en">Reader Information</xsd:documentation>
6373   </xsd:annotation>
6374   <xsd:sequence>
6375     <xsd:element name="readerEPC" type="epcglobal:EPC" minOccurs="0">
6376       <xsd:annotation>
6377         <xsd:documentation xml:lang="en">Reader EPC</xsd:documentation>
6378       </xsd:annotation>
6379     </xsd:element>
6380     <xsd:element name="readerName" type="xsd:string" minOccurs="0">
6381       <xsd:annotation>
6382         <xsd:documentation xml:lang="en">Reader name</xsd:documentation>
6383       </xsd:annotation>
6384     </xsd:element>
6385     <xsd:element name="readerHandle" type="xsd:int" minOccurs="0">
6386       <xsd:annotation>
6387         <xsd:documentation xml:lang="en">Reader handle</xsd:documentation>
6388       </xsd:annotation>
6389     </xsd:element>
6390     <xsd:element name="readerRole" type="xsd:string" minOccurs="0">
6391       <xsd:annotation>
6392         <xsd:documentation xml:lang="en">Reader role</xsd:documentation>
6393       </xsd:annotation>
6394     </xsd:element>
6395     <xsd:element name="readerTime" type="rm:readerTime" minOccurs="0">
6396       <xsd:annotation>
6397         <xsd:documentation xml:lang="en">Reader time</xsd:documentation>
6398       </xsd:annotation>
6399     </xsd:element>
6400   </xsd:sequence>
6401 </xsd:complexType>
6402 <!-- Alarms -->
6403 <!-- Alarm types (including their base types) are ordered in alphabetical order -->
6404 <xsd:complexType name="AlarmType" abstract="true">
6405   <xsd:annotation>
6406     <xsd:documentation xml:lang="en">Alarm is the base class for all of classes within the RMP object model that
6407     define the contents of alarm messages</xsd:documentation>
6408   </xsd:annotation>
6409   <xsd:complexContent>
6410     <xsd:extension base="epcglobal:Document">
6411       <xsd:sequence>
6412         <xsd:element name="id" type="xsd:string">
6413           <xsd:annotation>
6414             <xsd:documentation xml:lang="en">id of this alarm.</xsd:documentation>
6415           </xsd:annotation>
6416         </xsd:element>
6417         <xsd:element name="reader" type="rm:ReaderType" minOccurs="0">
6418           <xsd:annotation>
6419             <xsd:documentation xml:lang="en">Reader sending the alarm</xsd:documentation>
6420           </xsd:annotation>

```

```

6421     </xsd:element>
6422     <xsd:element name="name" type="xsd:string">
6423         <xsd:annotation>
6424             <xsd:documentation xml:lang="en">Type of Alarm message</xsd:documentation>
6425         </xsd:annotation>
6426     </xsd:element>
6427     <xsd:element name="level" type="rm:AlarmLevelParamType">
6428         <xsd:annotation>
6429             <xsd:documentation xml:lang="en">Severity level assigned to the alarm</xsd:documentation>
6430         </xsd:annotation>
6431     </xsd:element>
6432     <xsd:element name="suppressCount" type="xsd:int">
6433         <xsd:annotation>
6434             <xsd:documentation xml:lang="en">Number of times the generation of this Alarm has been
6435 suppressed</xsd:documentation>
6436         </xsd:annotation>
6437     </xsd:element>
6438     <xsd:any namespace="##any" processContents="lax">
6439         <xsd:annotation>
6440             <xsd:documentation>For standard and vendor extensions</xsd:documentation>
6441         </xsd:annotation>
6442     </xsd:any>
6443 </xsd:sequence>
6444 </xsd:extension>
6445 </xsd:complexContent>
6446 </xsd:complexType>
6447 <xsd:complexType name="FailedEraseAlarmType">
6448     <xsd:annotation>
6449         <xsd:documentation xml:lang="en">This alarm's receipt signals a tag erase failure.</xsd:documentation>
6450     </xsd:annotation>
6451     <xsd:complexContent>
6452         <xsd:extension base="rm:AlarmType">
6453             <xsd:sequence>
6454                 <xsd:element name="readPointName" type="xsd:string">
6455                     <xsd:annotation>
6456                         <xsd:documentation xml:lang="en">Read point where the erase failure
6457 occurred.</xsd:documentation>
6458                     </xsd:annotation>
6459                 </xsd:element>
6460                 <xsd:element name="failedEraseCount" type="xsd:int">
6461                     <xsd:annotation>
6462                         <xsd:documentation xml:lang="en">Value of AntennaReadPoint.FailedEraseCount element after the
6463 erase failure occurred.</xsd:documentation>
6464                     </xsd:annotation>
6465                 </xsd:element>
6466                 <xsd:element name="noiseLevel" type="xsd:int">
6467                     <xsd:annotation>
6468                         <xsd:documentation xml:lang="en">Value of the AntennaReadPoint.NoiseLevel element when the
6469 erase failure occurred.</xsd:documentation>
6470                     </xsd:annotation>
6471                 </xsd:element>
6472             </xsd:sequence>
6473         </xsd:extension>
6474     </xsd:complexContent>
6475 </xsd:complexType>
6476 <xsd:complexType name="FailedKillAlarmType">

```

```

6477 <xsd:annotation>
6478   <xsd:documentation xml:lang="en">This alarm's receipt signals a tag kill failure.</xsd:documentation>
6479 </xsd:annotation>
6480 <xsd:complexContent>
6481   <xsd:extension base="rm:AlarmType">
6482     <xsd:sequence>
6483       <xsd:element name="readPointName" type="xsd:string">
6484         <xsd:annotation>
6485           <xsd:documentation xml:lang="en">Read point where the kill failure occurred.</xsd:documentation>
6486         </xsd:annotation>
6487       </xsd:element>
6488       <xsd:element name="failedKillCount" type="xsd:int">
6489         <xsd:annotation>
6490           <xsd:documentation xml:lang="en">Value of AntennaReadPoint.FailedKillCount element after the kill
6491 failure occurred.</xsd:documentation>
6492         </xsd:annotation>
6493       </xsd:element>
6494       <xsd:element name="noiseLevel" type="xsd:int">
6495         <xsd:annotation>
6496           <xsd:documentation xml:lang="en">Value of the AntennaReadPoint.NoiseLevel element when the kill
6497 failure occurred.</xsd:documentation>
6498         </xsd:annotation>
6499       </xsd:element>
6500     </xsd:sequence>
6501   </xsd:extension>
6502 </xsd:complexContent>
6503 </xsd:complexType>
6504 <xsd:complexType name="FailedLockAlarmType">
6505   <xsd:annotation>
6506     <xsd:documentation xml:lang="en">This alarm's receipt signals a tag lock failure.</xsd:documentation>
6507   </xsd:annotation>
6508   <xsd:complexContent>
6509     <xsd:extension base="rm:AlarmType">
6510       <xsd:sequence>
6511         <xsd:element name="readPointName" type="xsd:string">
6512           <xsd:annotation>
6513             <xsd:documentation xml:lang="en">Read point where the lock failure occurred.</xsd:documentation>
6514           </xsd:annotation>
6515         </xsd:element>
6516         <xsd:element name="failedLockCount" type="xsd:int">
6517           <xsd:annotation>
6518             <xsd:documentation xml:lang="en">Value of AntennaReadPoint.FailedLockCount element after the
6519 lock failure occurred.</xsd:documentation>
6520           </xsd:annotation>
6521         </xsd:element>
6522         <xsd:element name="noiseLevel" type="xsd:int">
6523           <xsd:annotation>
6524             <xsd:documentation xml:lang="en">Value of the AntennaReadPoint.NoiseLevel element when the
6525 lock failure occurred.</xsd:documentation>
6526           </xsd:annotation>
6527         </xsd:element>
6528       </xsd:sequence>
6529     </xsd:extension>
6530   </xsd:complexContent>
6531 </xsd:complexType>
6532 <xsd:complexType name="FailedMemReadAlarmType">

```

```

6533     <xsd:annotation>
6534         <xsd:documentation xml:lang="en">This alarm's receipt signals a tag user-memory read
6535 failure.</xsd:documentation>
6536     </xsd:annotation>
6537     <xsd:complexContent>
6538         <xsd:extension base="rm:AlarmType">
6539             <xsd:sequence>
6540                 <xsd:element name="readPointName" type="xsd:string">
6541                     <xsd:annotation>
6542                         <xsd:documentation xml:lang="en">Read point where the memory read failure
6543 occurred.</xsd:documentation>
6544                     </xsd:annotation>
6545                 </xsd:element>
6546                 <xsd:element name="failedMemReadCount" type="xsd:int">
6547                     <xsd:annotation>
6548                         <xsd:documentation xml:lang="en">Value of AntennaReadPoint.FailedMemReadCount element after
6549 the memory read failure occurred.</xsd:documentation>
6550                     </xsd:annotation>
6551                 </xsd:element>
6552                 <xsd:element name="noiseLevel" type="xsd:int">
6553                     <xsd:annotation>
6554                         <xsd:documentation xml:lang="en">Value of the AntennaReadPoint.NoiseLevel element when the
6555 memory read failure occurred.</xsd:documentation>
6556                     </xsd:annotation>
6557                 </xsd:element>
6558             </xsd:sequence>
6559         </xsd:extension>
6560     </xsd:complexContent>
6561 </xsd:complexType>
6562 <xsd:complexType name="FailedWriteAlarmType">
6563     <xsd:annotation>
6564         <xsd:documentation xml:lang="en">This alarm's receipt signals a tag write failure.</xsd:documentation>
6565     </xsd:annotation>
6566     <xsd:complexContent>
6567         <xsd:extension base="rm:AlarmType">
6568             <xsd:sequence>
6569                 <xsd:element name="readPointName" type="xsd:string">
6570                     <xsd:annotation>
6571                         <xsd:documentation xml:lang="en">Read point where the write failure
6572 occurred.</xsd:documentation>
6573                     </xsd:annotation>
6574                 </xsd:element>
6575                 <xsd:element name="failedWriteCount" type="xsd:int">
6576                     <xsd:annotation>
6577                         <xsd:documentation xml:lang="en">Value of AntennaReadPoint.FailedWriteCount element after the
6578 write failure occurred.</xsd:documentation>
6579                     </xsd:annotation>
6580                 </xsd:element>
6581                 <xsd:element name="noiseLevel" type="xsd:int">
6582                     <xsd:annotation>
6583                         <xsd:documentation xml:lang="en">Value of the AntennaReadPoint.NoiseLevel element when the
6584 write failure occurred.</xsd:documentation>
6585                     </xsd:annotation>
6586                 </xsd:element>
6587             </xsd:sequence>
6588         </xsd:extension>

```

```

6589     </xsd:complexContent>
6590 </xsd:complexType>
6591 <xsd:complexType name="FreeMemoryAlarmType">
6592     <xsd:annotation>
6593         <xsd:documentation xml:lang="en">This alarm's receipt signals the movement of a reader device's free
6594 memory (represented in the abstract model by ReaderDevice.FreeMemory) below a specified threshold
6595 value.</xsd:documentation>
6596     </xsd:annotation>
6597     <xsd:complexContent>
6598         <xsd:extension base="rm:AlarmType">
6599             <xsd:sequence>
6600                 <xsd:element name="freeMemory" type="xsd:int">
6601                     <xsd:annotation>
6602                         <xsd:documentation xml:lang="en">Value of ReaderDevice.FreeMemory when the alarm was
6603 triggered.</xsd:documentation>
6604                     </xsd:annotation>
6605                 </xsd:element>
6606             </xsd:sequence>
6607         </xsd:extension>
6608     </xsd:complexContent>
6609 </xsd:complexType>
6610 <xsd:complexType name="IOPortOperStatusAlarmType">
6611     <xsd:annotation>
6612         <xsd:documentation xml:lang="en">This alarm's receipt signals a change in the operational status of a
6613 Reader's IO Port. </xsd:documentation>
6614     </xsd:annotation>
6615     <xsd:complexContent>
6616         <xsd:extension base="rm:TTOperStatusAlarmType">
6617             <xsd:sequence>
6618                 <xsd:element name="iOPortName" type="xsd:string">
6619                     <xsd:annotation>
6620                         <xsd:documentation xml:lang="en">IO port name.</xsd:documentation>
6621                     </xsd:annotation>
6622                 </xsd:element>
6623             </xsd:sequence>
6624         </xsd:extension>
6625     </xsd:complexContent>
6626 </xsd:complexType>
6627 <xsd:complexType name="NotificationChannelOperStatusAlarmType">
6628     <xsd:annotation>
6629         <xsd:documentation xml:lang="en">This alarm's receipt signals a change in the operational status of one of a
6630 Reader's notification channels.</xsd:documentation>
6631     </xsd:annotation>
6632     <xsd:complexContent>
6633         <xsd:extension base="rm:TTOperStatusAlarmType">
6634             <xsd:sequence>
6635                 <xsd:element name="notificationChannelName" type="xsd:string">
6636                     <xsd:annotation>
6637                         <xsd:documentation xml:lang="en"> Notification channel name.</xsd:documentation>
6638                     </xsd:annotation>
6639                 </xsd:element>
6640             </xsd:sequence>
6641         </xsd:extension>
6642     </xsd:complexContent>
6643 </xsd:complexType>
6644 <xsd:complexType name="ReaderDeviceOperStatusAlarmType">

```

```

6645     <xsd:annotation>
6646         <xsd:documentation xml:lang="en">This alarm's receipt signals a change in the operational status of a Reader.
6647     </xsd:documentation>
6648     </xsd:annotation>
6649     <xsd:complexContent>
6650         <xsd:extension base="rm:TTOperStatusAlarmType"/>
6651     </xsd:complexContent>
6652 </xsd:complexType>
6653 <xsd:complexType name="ReadPointOperStatusAlarmType">
6654     <xsd:annotation>
6655         <xsd:documentation xml:lang="en">This alarm's receipt signals a change in the operational status of one of a
6656 Reader's Read Points. </xsd:documentation>
6657     </xsd:annotation>
6658     <xsd:complexContent>
6659         <xsd:extension base="rm:TTOperStatusAlarmType">
6660             <xsd:sequence>
6661                 <xsd:element name="readPointName" type="xsd:string">
6662                     <xsd:annotation>
6663                         <xsd:documentation xml:lang="en"> Read point name.</xsd:documentation>
6664                     </xsd:annotation>
6665                 </xsd:element>
6666             </xsd:sequence>
6667         </xsd:extension>
6668     </xsd:complexContent>
6669 </xsd:complexType>
6670 <xsd:complexType name="SourceOperStatusAlarmType">
6671     <xsd:annotation>
6672         <xsd:documentation xml:lang="en">This alarm's receipt signals a change in the operational status of a logical
6673 source of EPC data on a Reader. </xsd:documentation>
6674     </xsd:annotation>
6675     <xsd:complexContent>
6676         <xsd:extension base="rm:TTOperStatusAlarmType">
6677             <xsd:sequence>
6678                 <xsd:element name="sourceName" type="xsd:string">
6679                     <xsd:annotation>
6680                         <xsd:documentation xml:lang="en"> Source name.</xsd:documentation>
6681                     </xsd:annotation>
6682                 </xsd:element>
6683             </xsd:sequence>
6684         </xsd:extension>
6685     </xsd:complexContent>
6686 </xsd:complexType>
6687 <xsd:complexType name="TTOperStatusAlarmType" abstract="true">
6688     <xsd:annotation>
6689         <xsd:documentation xml:lang="en">Base class for all transition triggered Operational Status Alarms.
6690 </xsd:documentation>
6691     </xsd:annotation>
6692     <xsd:complexContent>
6693         <xsd:extension base="rm:AlarmType">
6694             <xsd:sequence>
6695                 <xsd:element name="fromState" type="rm:OperationalStatusParamType">
6696                     <xsd:annotation>
6697                         <xsd:documentation xml:lang="en">From state of the transition.</xsd:documentation>
6698                     </xsd:annotation>
6699                 </xsd:element>
6700                 <xsd:element name="toState" type="rm:OperationalStatusParamType">

```

```

6701         <xsd:annotation>
6702             <xsd:documentation xml:lang="en">To state of the transition.</xsd:documentation>
6703         </xsd:annotation>
6704     </xsd:element>
6705 </xsd:sequence>
6706 </xsd:extension>
6707 </xsd:complexContent>
6708 </xsd:complexType>
6709 </xsd:schema>
6710

```

## 6711 10.3.4 Common Data Formats

```

6712 <?xml version="1.0" encoding="UTF-8"?>
6713 <xsd:schema targetNamespace="urn:epcglobal:rm:xsd:1" elementFormDefault="unqualified"
6714 attributeFormDefault="unqualified" version="1.0" xmlns:rm="urn:epcglobal:rm:xsd:1"
6715 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
6716     <xsd:annotation>
6717         <xsd:documentation xml:lang="en">
6718             <xsd:documentation xml:lang="en">
6719                 </xsd:documentation>
6720             </xsd:documentation>
6721             <!-- Reader Management types -->
6722             <!-- types are ordered in alphabetical order -->
6723         <xsd:annotation>
6724             <xsd:documentation xml:lang="en"> Reader management common types. All types end with the
6725                 suffix "Type". All the command parameter or reply return value reference types are
6726                 suffixed with "ParamType". All named objects are referenced using "name". For example,
6727                 NotificationChannelParamType contains a reference (its name) instead of actual notification channel
6728                 itself.</xsd:documentation>
6729         </xsd:annotation>
6730         <xsd:simpleType name="AddressParamType">
6731             <xsd:annotation>
6732                 <xsd:documentation xml:lang="en">The address for a channel</xsd:documentation>
6733             </xsd:annotation>
6734             <xsd:restriction base="xsd:anyURI"/>
6735         </xsd:simpleType>
6736         <xsd:simpleType name="AdministrativeStatusParamType">
6737             <xsd:annotation>
6738                 <xsd:documentation xml:lang="en"> administrative status</xsd:documentation>
6739             </xsd:annotation>
6740             <xsd:restriction base="xsd:string">
6741                 <xsd:enumeration value="UP"/>
6742                 <xsd:enumeration value="DOWN"/>
6743             </xsd:restriction>
6744         </xsd:simpleType>
6745         <xsd:complexType name="AlarmChannelListParamType">
6746             <xsd:annotation>
6747                 <xsd:documentation> A list of alarm channels</xsd:documentation>
6748             </xsd:annotation>
6749             <xsd:sequence>
6750                 <xsd:element name="list">
6751                     <xsd:complexType>
6752                         <xsd:sequence>
6753                             <xsd:element name="value" type="rm:AlarmChannelParamType" maxOccurs="unbounded"/>

```

```

6754         </xsd:sequence>
6755     </xsd:complexType>
6756 </xsd:element>
6757 </xsd:sequence>
6758 </xsd:complexType>
6759 <xsd:simpleType name="AlarmChannelParamType">
6760     <xsd:annotation>
6761         <xsd:documentation xml:lang="en">The name of a alarm channel.</xsd:documentation>
6762     </xsd:annotation>
6763     <xsd:restriction base="xsd:string"/>
6764 </xsd:simpleType>
6765 <xsd:simpleType name="AlarmControlParamType">
6766     <xsd:annotation>
6767         <xsd:documentation xml:lang="en">The name of an alarm control.</xsd:documentation>
6768     </xsd:annotation>
6769     <xsd:restriction base="xsd:string"/>
6770 </xsd:simpleType>
6771 <xsd:simpleType name="AlarmLevelParamType">
6772     <xsd:annotation>
6773         <xsd:documentation xml:lang="en"> Alarm level</xsd:documentation>
6774     </xsd:annotation>
6775     <xsd:restriction base="xsd:string">
6776         <xsd:enumeration value="EMERGENCY"/>
6777         <xsd:enumeration value="ALERT"/>
6778         <xsd:enumeration value="CRITICAL"/>
6779         <xsd:enumeration value="ERROR"/>
6780         <xsd:enumeration value="WARNING"/>
6781         <xsd:enumeration value="NOTICE"/>
6782         <xsd:enumeration value="INFORMATIONAL"/>
6783         <xsd:enumeration value="DEBUG"/>
6784     </xsd:restriction>
6785 </xsd:simpleType>
6786 <xsd:simpleType name="EdgeTriggeredAlarmControlParamType">
6787     <xsd:annotation>
6788         <xsd:documentation xml:lang="en">The name of an edge triggered alarm control.</xsd:documentation>
6789     </xsd:annotation>
6790     <xsd:restriction base="rm:AlarmControlParamType"/>
6791 </xsd:simpleType>
6792 <xsd:simpleType name="EdgeTriggeredAlarmDirectionParamType">
6793     <xsd:annotation>
6794         <xsd:documentation xml:lang="en"> Edge triggered alarm direction.</xsd:documentation>
6795     </xsd:annotation>
6796     <xsd:restriction base="xsd:string">
6797         <xsd:enumeration value="RISING"/>
6798         <xsd:enumeration value="FALLING"/>
6799     </xsd:restriction>
6800 </xsd:simpleType>
6801 <xsd:simpleType name="EdgeTriggeredAlarmStatusParamType">
6802     <xsd:annotation>
6803         <xsd:documentation xml:lang="en"> Edge triggered alarm status.</xsd:documentation>
6804     </xsd:annotation>
6805     <xsd:restriction base="xsd:string">
6806         <xsd:enumeration value="ARMED"/>
6807         <xsd:enumeration value="FIRED"/>
6808     </xsd:restriction>
6809 </xsd:simpleType>

```



```

6810 <xsd:simpleType name="IOPortParamType">
6811   <xsd:annotation>
6812     <xsd:documentation xml:lang="en">The name of an IO port.</xsd:documentation>
6813   </xsd:annotation>
6814   <xsd:restriction base="xsd:string"/>
6815 </xsd:simpleType>
6816 <xsd:simpleType name="OperationalStatusParamType">
6817   <xsd:annotation>
6818     <xsd:documentation xml:lang="en"> operational status</xsd:documentation>
6819   </xsd:annotation>
6820   <xsd:restriction base="xsd:string">
6821     <xsd:enumeration value="UNKNOWN"/>
6822     <xsd:enumeration value="UP"/>
6823     <xsd:enumeration value="DOWN"/>
6824     <xsd:enumeration value="OTHER"/>
6825     <xsd:enumeration value="ANY"/>
6826   </xsd:restriction>
6827 </xsd:simpleType>
6828 <xsd:complexType name="readerTime">
6829   <xsd:annotation>
6830     <xsd:documentation xml:lang="en">Time format</xsd:documentation>
6831   </xsd:annotation>
6832   <xsd:sequence>
6833     <xsd:element name="readerNowTick" type="xsd:string" minOccurs="0">
6834       <xsd:annotation>
6835         <xsd:documentation xml:lang="en">time in ticks</xsd:documentation>
6836       </xsd:annotation>
6837     </xsd:element>
6838     <xsd:element name="readerNowUTC" type="xsd:dateTime" minOccurs="0">
6839       <xsd:annotation>
6840         <xsd:documentation xml:lang="en"> UTC time</xsd:documentation>
6841       </xsd:annotation>
6842     </xsd:element>
6843   </xsd:sequence>
6844 </xsd:complexType>
6845 <xsd:simpleType name="TTOperationalStatusAlarmControlParamType">
6846   <xsd:annotation>
6847     <xsd:documentation xml:lang="en">The name of a transition triggered status alarm
6848 control.</xsd:documentation>
6849   </xsd:annotation>
6850   <xsd:restriction base="rm:AlarmControlParamType"/>
6851 </xsd:simpleType>
6852 </xsd:schema>

```

### 6853 10.3.5 EPCglobal Standard Header

```

6854 <?xml version="1.0" encoding="UTF-8"?>
6855 <xsd:schema xmlns:epcglobal="urn:epcglobal:xsd:1" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6856 targetNamespace="urn:epcglobal:xsd:1" elementFormDefault="unqualified" attributeFormDefault="unqualified"
6857 version="1.0">
6858   <xsd:annotation>
6859     <xsd:documentation>
6860       <epcglobal:copyright>Copyright&#169;2005-2006 Epcglobal Inc., All Rights Reserved.</epcglobal:copyright>
6861       <epcglobal:disclaimer>EPCglobal Inc., its members, officers, directors, employees, or agents shall not be liable
6862 for any injury, loss, damages, financial or otherwise, arising from, related to, or caused by the use of this document.
6863 The use of said document shall constitute your express consent to the foregoing exculpation.</epcglobal:disclaimer>

```

6864  
6865  
6866  
6867  
6868  
6869  
6870  
6871  
6872  
6873  
6874  
6875  
6876  
6877  
6878  
6879  
6880  
6881  
6882  
6883  
6884  
6885  
6886  
6887  
6888  
6889  
6890  
6891  
6892  
6893  
6894  
6895  
6896  
6897  
6898  
6899  
6900  
6901  
6902  
6903  
6904  
6905  
6906  
6907  
6908  
6909  
6910  
6911  
6912

```
<epcglobal:specification>EPCglobal common components Version 1.0</epcglobal:specification>
  </xsd:documentation>
</xsd:annotation>
<xsd:complexType name="Document" abstract="true">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      EPCglobal document properties for all messages.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:attribute name="schemaVersion" type="xsd:decimal" use="required">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
        The version of the schema corresponding to which the instance conforms.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
  <xsd:attribute name="creationDate" type="xsd:dateTime" use="required">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
        The date the message was created. Used for auditing and logging.
      </xsd:documentation>
    </xsd:annotation>
  </xsd:attribute>
</xsd:complexType>
<xsd:complexType name="EPC">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      EPC represents the Electronic Product Code.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleContent>
    <xsd:extension base="xsd:string"/>
  </xsd:simpleContent>
</xsd:complexType>
</xsd:schema>

  </xsd:attribute>
</xsd:complexType>
<xsd:complexType name="EPC">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      EPC represents the Electronic Product Code.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleContent>
    <xsd:extension base="xsd:string"/>
  </xsd:simpleContent>
</xsd:complexType>
</xsd:schema>
```

6913  
6914  
6915

## 10.4 SNMP MIB

This specification defines two MIB modules in the EPCglobal enterprise tree that expose the EPCglobal Reader Object Model across an SNMP protocol interface.

6916 The first of these MIB modules, EPCGLOBAL-SMI-MIB, is the EPCglobal central  
 6917 registration module. It contains the specification of the private enterprise number  
 6918 (22695) the Internet Assigned Numbers Authority (IANA) assigned to EPCglobal. In  
 6919 addition, it defines the top-level organization of the EPCglobal private enterprise  
 6920 namespace.

6921

6922 The second MIB module, EPCGLOBAL-READER-MIB, contains MIB definitions  
 6923 corresponding to objects within the Reader Object Model. This MIB defines model  
 6924 objects that are particular to the health monitoring of RFID Readers.

6925 When there is a clear mapping of Reader Object Model elements to standard MIB-II  
 6926 object definitions, we rely on the IETF-standardized MIB modules to expose these model  
 6927 elements across SNMP, rather than duplicating these object definitions within EPCglobal  
 6928 MIB modules. Below is a table displaying the mapping of selected Reader Object Model  
 6929 elements to object definitions within IETF MIB modules (in particular, SNMPv2-MIB  
 6930 defined in RFC3418/STD 62 and SNMP-TARGET-MIB, RFC3413/STD62).

<b>Reader Object Model element</b>	<b>IETF-Defined MIB Module</b>	<b>MIB OID</b>
ReaderDevice.Manufacturer ReaderDevice.Model ReaderDevice.ManufacturerDescription	SNMPv2-MIB	sysDescr
ReaderDevice.LocationDescription	SNMPv2-MIB	location
ReaderDevice.Contact	SNMPv2-MIB	sysContact
ReaderDevice.TimeTicks	SNMPv2-MIB	sysUpTime
ReaderDevice.AlarmChannels	SNMP-TARGET-MIB	snmpTargetObjects
AlarmChannel.Name	SNMP-TARGET-MIB	snmpTargetAddrName
AlarmChannel.Address	SNMP-TARGET-MIB	snmpTargetAddrTDomain snmpTargetAddrTAddress
Alarm.TimeTicks	Protocol Operations for SNMP (RFC3416)	sysUpTime

6931

6932

6933 In addition to compliance with the EPC Global Reader Management MIB, a compliant  
 6934 system that supports the SNMP binding SHALL implement support for the following  
 6935 IETF-standardized MIB groups:

6936 • the MIB-II System Group defined in the SNMPv2-MIB module, defined in RFC  
6937 3418,

6938 • the MIB-II IP Group, defined in the IP-MIB module, RFC 2011,

6939 • the MIB-II Interfaces Group, defined in IF-MIB module, RFC 2863

6940 These groups define the basic System Identification operations, Internet Protocol and  
6941 Network Interfaces monitoring objects and operations. In order to avoid unnecessary  
6942 duplication and to harmonize reader management, as a networked device, with that of  
6943 other SNMP-manageable devices, support of these MIB-II groups is chosen in place of  
6944 defining these objects within in the EPCglobal RFID Reader Management MIB.

6945 Protocol support for the management of Alarm Channels is optional; however, if a reader  
6946 vendor chooses to support SNMP-based management of Alarm Channels, the vendor  
6947 SHALL do so by implementing the SNMP-TARGET-MIB module, defined in RFC  
6948 3413.

6949 Within the EPCglobal MIB structure, elements referred to as "Alarms" in the Abstract  
6950 Model are referred to as "Notifications". This is because the term "Alarm" in SNMP is  
6951 used as a term of art (for example, in RFC 3877 and RFC 3878) and could lead to  
6952 confusion within existing SNMP installations.

6953 Alarm TimeTicks are included in the SNMP Notification data structure mandated by  
6954 RFC 3416, therefore not explicitly included in the MIB.

6955 Unless otherwise specified within the MIB, vendors MAY implement index persistence  
6956 for any of the EPCglobal defined MIB structures. This statement only applies to the  
6957 EPCglobal enterprise space.

#### 6958 **10.4.1 Vendor Extension Details**

6959 Vendors wishing to provide vendor-specific extensions in SNMP MAY provide support  
6960 for MIB modules outside the EPCglobal MIB space. For example, if a vendor supports  
6961 vendor-specific MIB on their devices, they MAY also support these MIB modules on  
6962 EPCglobal compliant readers. Additionally, a EPCglobal compliant readers MAY  
6963 provide support for any IETF-standard MIB module, such as the HOST-RESOURCES-  
6964 MIB (RFC 2790).



6966 **10.4.2 EPCglobal RFID Reader Management MIB**

6967 **10.4.2.1 EPCglobal SMI MIB**

```
6968 -- *****
6969 -- Copyright (c) 2005-2007 EPCglobal Inc(tm), All Rights Reserved.
6970 -- *****
6971 EPCGLOBAL-SMI-MIB DEFINITIONS ::= BEGIN
6972
6973 IMPORTS
6974     enterprises,
6975     MODULE-IDENTITY
6976     FROM SNMPv2-SMI;
6977
6978
6979 epcglobal MODULE-IDENTITY
6980     LAST-UPDATED "200610040000Z"
6981     ORGANIZATION "EPCglobal, Inc."
6982     CONTACT-INFO
6983         "
6984             EPCglobal MIB Administrator
6985             GS1/EPCglobal, Inc.
6986             Princeton Pike Corporate Center
6987             1009 Lenox Drive, Suite 202
6988             Lawrenceville, NJ 08648
6989             US
6990
6991             Tel: +1 609 620 0200
6992             Email: mibs@lists.epcglobalinc.org"
6993
6994 DESCRIPTION
6995     "The EPCglobal central registration module, containing
6996     the top-level organization of the EPCglobal private
6997     enterprise namespace."
6998
6999 REVISION      "200610040000Z"
7000
7001 DESCRIPTION
7002     "Defined in conformance with the EPCglobal Reader Management and
7003     Reader Protocol specifications"
7004
7005 ::= {enterprises 22695} -- assigned by IANA
7006
7007 --
7008 -- The EPCglobal private enterprise number assigned by
7009 -- the Internet Assigned Numbers Authority (IANA).
7010 --
7011 epcgSmiManagement      OBJECT IDENTIFIER ::= { epcglobal 1 }
7012 epcgSmiExperimental    OBJECT IDENTIFIER ::= { epcglobal 2 }
7013
7014 END
```

7013 **10.4.2.2 EPCglobal Reader MIB**

```
7014 -- Copyright (c) 2005, 2007 EPCglobal Inc(tm), All Rights Reserved.
7015 EPCGLOBAL-READER-MIB DEFINITIONS ::= BEGIN
7016
7017 IMPORTS
7018     Gauge32,
7019     Integer32,
7020     MODULE-IDENTITY,
7021     NOTIFICATION-TYPE,
7022     OBJECT-TYPE,
```

```

7023 Unsigned32,
7024 Counter32
7025     FROM SNMPv2-SMI
7026
7027 SnmpAdminString
7028     FROM SNMP-FRAMEWORK-MIB
7029
7030 DateAndTime,
7031 RowPointer,
7032 RowStatus,
7033 TEXTUAL-CONVENTION,
7034 TruthValue
7035     FROM SNMPv2-TC
7036
7037 MODULE-COMPLIANCE,
7038 NOTIFICATION-GROUP,
7039 OBJECT-GROUP
7040     FROM SNMPv2-CONF
7041
7042 InetAddress,
7043 InetAddressType
7044     FROM INET-ADDRESS-MIB
7045
7046 sysName
7047     FROM SNMPv2-MIB
7048
7049 epcgSmiManagement
7050     FROM EPCGLOBAL-SMI-MIB;
7051
7052 epcgReaderMIB MODULE-IDENTITY
7053     LAST-UPDATED "200703080000Z"
7054     ORGANIZATION "EPCglobal, Inc."
7055     CONTACT-INFO
7056         "      EPCglobal MIB Administrator
7057             GS1/EPCglobal, Inc.
7058             Princeton Pike Corporate Center
7059             1009 Lenox Drive, Suite 202
7060             Lawrenceville, NJ 08648
7061             US
7062
7063             Tel:   +1 609 620 0200
7064             Email: mibs@lists.epcglobalinc.org"
7065     DESCRIPTION
7066         "The MIB Module describing an EPCglobal compliant RFID reader."
7067
7068     REVISION      "200703080000Z"
7069
7070     DESCRIPTION
7071         "Defined in conformance with the EPCglobal Reader Management and
7072         Reader Protocol specifications.
7073
7074         Abbreviations:
7075
7076         DHCP   - Dynamic Host Configuration Protocol
7077         DNS     - Domain Name System
7078         EPC     - Electronic Product Code
7079         URI     - Uniform Resource Identifier
7080         RFID   - Radio Frequency Identification
7081         UTC     - Coordinated Universal Time
7082
7083         Reference EPCglobal Reader Protocol Specification 1.1"
7084

```

```

7085 ::= {epcgSmiManagement 1}
7086
7087 --*****
7088 --** The EPCglobal Reader MIB module sub-trees
7089 --*****
7090 epcgReaderNotifs OBJECT IDENTIFIER ::= { epcgReaderMIB 0 }
7091 epcgReaderObjects OBJECT IDENTIFIER ::= { epcgReaderMIB 1 }
7092 epcgReaderConformance OBJECT IDENTIFIER ::= { epcgReaderMIB 2 }
7093 epcgReaderDevice OBJECT IDENTIFIER ::= { epcgReaderObjects 1 }
7094 epcgReadPoints OBJECT IDENTIFIER ::= { epcgReaderObjects 2 }
7095 epcgAntennaReadPoints OBJECT IDENTIFIER ::= { epcgReaderObjects 3 }
7096 epcgIoPorts OBJECT IDENTIFIER ::= { epcgReaderObjects 4 }
7097 epcgSources OBJECT IDENTIFIER ::= { epcgReaderObjects 5 }
7098 epcgNotificationChannels OBJECT IDENTIFIER ::= { epcgReaderObjects 6 }
7099 epcgTriggers OBJECT IDENTIFIER ::= { epcgReaderObjects 7 }
7100
7101
7102 --*****
7103 --** Textual Conventions used by EPCglobal Reader MIB modules **
7104 --*****
7105
7106 EpcgCurrentCountSinceReset ::= TEXTUAL-CONVENTION
7107     STATUS current
7108     DESCRIPTION
7109         "A counter that always increments until it is reset."
7110     SYNTAX Gauge32
7111
7112 EpcgNotifLevel ::= TEXTUAL-CONVENTION
7113     STATUS current
7114     DESCRIPTION
7115         "Severity level of an notification."
7116     SYNTAX INTEGER {
7117         emergency (0),
7118         alert (1),
7119         critical (2),
7120         error (3),
7121         warning (4),
7122         notice (5),
7123         informational (6),
7124         debug (7)
7125     }
7126
7127 EpcgOperationalEnable ::= TEXTUAL-CONVENTION
7128     STATUS current
7129     DESCRIPTION
7130         "Each bit represent whether a particular state defined
7131         by the EpcgOperationalStatus is enable for notifications."
7132     SYNTAX BITS { unknown(0),
7133         other (1),
7134         up (2),
7135         down (3)
7136     }
7137
7138 EpcgOperationalStatus ::= TEXTUAL-CONVENTION
7139     STATUS current
7140     DESCRIPTION
7141         "A value representing operational status."
7142     SYNTAX INTEGER { unknown (1),
7143         other (2),
7144         up (3),
7145         down (4)
7146     }

```



```

7147
7148 EpcgAdministrativeStatus ::= TEXTUAL-CONVENTION
7149     STATUS          current
7150     DESCRIPTION
7151         "A value representing administrative status."
7152     SYNTAX INTEGER { up (3),
7153                 down (4)
7154                 }
7155
7156 EpcgThreshold ::= TEXTUAL-CONVENTION
7157     STATUS          current
7158     DESCRIPTION
7159         "This textual convention defines the allowed values
7160         used to define a threshold."
7161     SYNTAX Unsigned32 (0..4294967295)
7162
7163 EpcgTriggerType ::= TEXTUAL-CONVENTION
7164     STATUS          current
7165     DESCRIPTION "The types of allowed triggers."
7166     SYNTAX INTEGER { none (1),
7167                 timer (2),
7168                 continuous (3),
7169                 ioEdge (4),
7170                 vendorExtension (5),
7171                 ioValue (6)
7172                 }
7173
7174
7175
7176 --*****
7177 --** Reader Notifications **
7178 --*****
7179
7180 epcgReaderDeviceOperationState NOTIFICATION-TYPE
7181     OBJECTS { sysName,
7182             epcgRdrDevTimeUtc,
7183             epcgRdrDevOperNotifStateLevel,
7184             epcgRdrDevOperStatusPrior,
7185             epcgRdrDevOperStatus
7186             }
7187     STATUS current
7188     DESCRIPTION
7189         "This notification is generated when a reader device
7190         undergoes a change in operational status. In some
7191         environment certain states may be transitory and
7192         do not need to generate notifications. In addition,
7193         a device may cycle between two states. Therefore,
7194         notifications will be generated whenever the state
7195         transitions into a state enabled for notifications
7196         as defined by the epcgRdrDevOperNotifToState
7197         object. Also, notifications will be generate whenever
7198         the state transitions out of a state define by the
7199         epcgRdrDevOperNotifFromState object. The
7200         epcgRdrDevOperStateSuppressInterval object can be
7201         configured to suppress the excessive generation of
7202         notifications."
7203     ::= { epcgReaderNotifs 1 }
7204
7205 epcgRdrDevMemoryState NOTIFICATION-TYPE
7206     OBJECTS { sysName,
7207             epcgRdrDevTimeUtc,
7208             epcgRdrDevFreeMemoryNotifLevel,

```

```

7209         epcgRdrDevFreeMemory
7210     }
7211 STATUS current
7212 DESCRIPTION
7213     "This notification is generated when the reader's
7214     free memory state changes and the following transitions
7215     are possible
7216
7217     normal -> shortage
7218
7219     A memory shortage is detected when the epcgRdrDevFreeMemory
7220     object changes to a value that is equal to or below the
7221     value specified by the epcgRdrDevFreeMemoryOnsetThreshold
7222     object the object.
7223
7224     shortage -> normal
7225
7226     A memory shortage ends when the epcgRdrDevFreeMemory
7227     object changes to a value that is equal to or greater than
7228     the value specified by the
7229     epcgRdrDevFreeMemoryAbateThreshold object.
7230
7231     The epcgRdrDevFreeMemoryOnsetThreshold and
7232     epcgRdrDevFreeMemoryAbateThreshold threshold will be
7233     different for each platform and should be related to the
7234     amount of available memory. For example, a device may
7235     decided to set the onset threshold to 20 percent of
7236     total memory and the abate to 35 percent of total memory.
7237
7238     The value of the epcgRdrDevFreeMemoryOnsetThreshold
7239     object must be less than the value specified for the
7240     epcgRdrDevFreeMemoryOnsetAbate object. In addition,
7241     the difference between these object should be sufficient
7242     to prevent excessive number of notification being
7243     generated when small amounts of memory are used then
7244     freed. The epcgRdrDevMemStateSuppressInterval object can
7245     be configured to suppress the excessive generation of
7246     notifications."
7247     ::= { epcgReaderNotifs 2 }
7248
7249 epcgReadPointOperationState NOTIFICATION-TYPE
7250     OBJECTS { sysName,
7251         epcgRdrDevTimeUtc,
7252         epcgReadPointOperNotifyStateLevel,
7253         epcgReadPointName,
7254         epcgReadPointOperStatusPrior,
7255         epcgReadPointOperStatus
7256     }
7257 STATUS current
7258 DESCRIPTION
7259     "This notification is generated when a read point
7260     undergoes a change in operational status. In some
7261     environment certain states may be transitory and
7262     do not need to generate notifications. In addition,
7263     a device may cycle between two states. Therefore,
7264     notifications will be generated whenever the state
7265     transitions into a state enabled for notifications
7266     as defined by the epcgRdrDevOperNotifToState
7267     object. Also, notifications will be generate whenever
7268     the state transitions out of a state define by the
7269     epcgRdrDevOperNotifFromState object. The
7270     epcgReadPointOperStateSuppressInterval object can be

```

```

7271         configured to suppress the excessive generation of
7272         notifications."
7273         ::= { epcgReaderNotifs 3 }
7274
7275 epcgReaderAntennaReadFailure NOTIFICATION-TYPE
7276     OBJECTS { sysName,
7277               epcgRdrDevTimeUtc,
7278               epcgAntRdPntReadFailureNotifLevel,
7279               epcgReadPointName,
7280               epcgAntRdPntMemoryReadFailures,
7281               epcgAntRdPntNoiseLevel
7282             }
7283     STATUS current
7284     DESCRIPTION
7285         "This notification is generated when a memory read operation
7286         across an antenna fails. To prevent an excessive number
7287         of notifications the epcgAntRdPntSuppressInterval can be
7288         configured to suppress generation."
7289     ::= { epcgReaderNotifs 4 }
7290
7291 epcgReaderAntennaWriteFailure NOTIFICATION-TYPE
7292     OBJECTS { sysName,
7293               epcgRdrDevTimeUtc,
7294               epcgAntRdPntWriteFailuresNotifLevel,
7295               epcgReadPointName,
7296               epcgAntRdPntWriteFailures,
7297               epcgAntRdPntNoiseLevel
7298             }
7299     STATUS current
7300     DESCRIPTION
7301         "This notification is generated when a write operation
7302         across an antenna fails. To prevent an excessive number
7303         of notifications the epcgAntRdPntSuppressInterval can be
7304         configured to suppress generation"
7305     ::= { epcgReaderNotifs 5 }
7306
7307 epcgReaderAntennaKillFailure NOTIFICATION-TYPE
7308     OBJECTS { sysName,
7309               epcgRdrDevTimeUtc,
7310               epcgAntRdPntKillFailuresNotifLevel,
7311               epcgReadPointName,
7312               epcgAntRdPntKillFailures,
7313               epcgAntRdPntNoiseLevel
7314             }
7315     STATUS current
7316     DESCRIPTION
7317         "This notification is generated when a kill operation
7318         across an antenna fails. To prevent an excessive number
7319         of notifications the epcgAntRdPntSuppressInterval can be
7320         configured to suppress generation"
7321     ::= { epcgReaderNotifs 6 }
7322
7323 epcgReaderAntennaEraseFailure NOTIFICATION-TYPE
7324     OBJECTS { sysName,
7325               epcgRdrDevTimeUtc,
7326               epcgAntRdPntEraseFailuresNotifLevel,
7327               epcgReadPointName,
7328               epcgAntRdPntEraseFailures,
7329               epcgAntRdPntNoiseLevel
7330             }
7331     STATUS current
7332     DESCRIPTION

```

```

7333         "This notification is generated when an erase operation
7334         across an antenna fails. To prevent an excessive number
7335         of notifications the epcgAntRdPntSuppressInterval can be
7336         configured to suppress generation"
7337         ::= { epcgReaderNotifs 7 }
7338
7339 epcgReaderAntennaLockFailure NOTIFICATION-TYPE
7340     OBJECTS { sysName,
7341               epcgRdrDevTimeUtc,
7342               epcgAntRdPntLockFailuresNotifLevel,
7343               epcgReadPointName,
7344               epcgAntRdPntLockFailures,
7345               epcgAntRdPntNoiseLevel
7346             }
7347     STATUS current
7348     DESCRIPTION
7349         "This notification is generated when a lock operation
7350         across an antenna fails. To prevent an excessive number
7351         of notifications the epcgAntRdPntSuppressInterval can be
7352         configured to suppress generation"
7353         ::= { epcgReaderNotifs 8 }
7354
7355 epcgReaderIoPortOperationState NOTIFICATION-TYPE
7356     OBJECTS { sysName,
7357               epcgRdrDevTimeUtc,
7358               epcgIoPortOperStatusNotifLevel,
7359               epcgIoPortName,
7360               epcgIoPortOperStatusPrior,
7361               epcgIoPortOperStatus
7362             }
7363     STATUS current
7364     DESCRIPTION
7365         "This notification is generated when an IO port
7366         undergoes a change in operation status that falls
7367         within the set of notification-triggering state transitions
7368         specified by the epcgIoPortOperStatusNotifFromState and
7369         epcgIoPortOperStatusNotifToState object values. The
7370         epcgIoPortOperStateSuppressInterval object can be
7371         configured to suppress the excessive generation of
7372         notifications."
7373         ::= { epcgReaderNotifs 9 }
7374
7375 epcgReaderSourceOperationState NOTIFICATION-TYPE
7376     OBJECTS { sysName,
7377               epcgRdrDevTimeUtc,
7378               epcgSrcOperStatusNotifyLevel,
7379               epcgSrcName,
7380               epcgSrcOperStatusPrior,
7381               epcgSrcOperStatus
7382             }
7383     STATUS current
7384     DESCRIPTION
7385         "This notification is generated when a source
7386         undergoes a change in operational status that
7387         falls within the set of notification-triggering state
7388         transitions specified by the epcgSrcOperStatusNotifFromState
7389         and epcgSrcOperStatusNotifToState object values. The
7390         notifications may be generated by a from state, a to state,
7391         or some combination of from and to states. The
7392         epcgRdrSourceOperStateSuppressInterval object can be
7393         configured to suppress the excessive generation of
7394         notifications."

```

```

7395 ::= { epcgReaderNotifs 10 }
7396
7397 epcgReaderNotificationChanOperState NOTIFICATION-TYPE
7398 OBJECTS { sysName,
7399         epcgRdrDevTimeUtc,
7400         epcgNotifChanOperNotifLevel,
7401         epcgNotifChanName,
7402         epcgNotifChanOperStatusPrior,
7403         epcgNotifChanOperStatus
7404     }
7405 STATUS current
7406 DESCRIPTION
7407     "This notification is generated when a notification
7408     channel undergoes a change in operational status that
7409     falls within the set of notification-triggering state
7410     transitions specified by the
7411     epcgNotifChanOperNotifFromState and
7412     epcgNotifChanOperNotifToState object values. The
7413     notifications may be generated by a from state, a to state, or some
7414     combination of from and to states. To prevent an excessive number
7415     of notifications the epcgRdrChanOperStateSuppressInterval can be
7416     configured to suppress generation"
7417 ::= { epcgReaderNotifs 11 }
7418
7419
7420 --*****
7421 --** Reader Device Information Objects **
7422 --*****
7423 epcgReaderDeviceInformation
7424     OBJECT IDENTIFIER ::= { epcgReaderDevice 1 }
7425
7426 -- Many of the objects contained within the ReaderDevice class of the
7427 -- EPCglobal Reader Management abstract model are represented, within
7428 -- the SNMP binding, by objects within in the System group (SNMPv2-MIB,
7429 -- see RFC 3418). In particular,
7430 --
7431 --     o System group's sysName object represents ReaderDevice.Name
7432 --     o System group's sysDescr object represents ReaderDevice.Manufacturer
7433 --     and ReaderDevice.Model
7434 --     o System group's sysLocation object represents
7435 --     ReaderDevice.LocationDescription
7436 --     o System group's sysContact object represents ReaderDevice.Contact
7437 --     o System group's sysUpTime object represents ReaderDevice.TimeTicks
7438 --
7439 epcgRdrDevDescription OBJECT-TYPE
7440     SYNTAX                SnmpAdminString
7441     MAX-ACCESS             read-only
7442     STATUS                 current
7443     DESCRIPTION
7444         "The operator's textual description of the reader."
7445     ::= { epcgReaderDeviceInformation 1 }
7446
7447 epcgRdrDevRole OBJECT-TYPE
7448     SYNTAX                SnmpAdminString
7449     MAX-ACCESS             read-only
7450     STATUS                 current
7451     DESCRIPTION
7452         "Reader functional role; e.g., Pallet Verification Portal."
7453     ::= { epcgReaderDeviceInformation 2 }
7454
7455 epcgRdrDevEpc OBJECT-TYPE
7456     SYNTAX                SnmpAdminString

```

```

7457 MAX-ACCESS          read-only
7458 STATUS             current
7459 DESCRIPTION
7460     "Manufacturer assigned EPC URI."
7461     ::= { epcgReaderDeviceInformation 3 }
7462
7463 epcgRdrDevSerialNumber OBJECT-TYPE
7464 SYNTAX              SnmpAdminString
7465 MAX-ACCESS          read-only
7466 STATUS             current
7467 DESCRIPTION
7468     "Manufacturer assigned serial number."
7469     ::= { epcgReaderDeviceInformation 4 }
7470
7471 epcgRdrDevTimeUtc   OBJECT-TYPE
7472 SYNTAX              DateAndTime
7473 MAX-ACCESS          read-only
7474 STATUS             current
7475 DESCRIPTION
7476     "Reader device wall clock time in UTC."
7477     ::= { epcgReaderDeviceInformation 5 }
7478
7479 epcgRdrDevCurrentSource OBJECT-TYPE
7480 SYNTAX              RowPointer
7481 MAX-ACCESS          read-only
7482 STATUS             current
7483 DESCRIPTION
7484     "A pointer to the row in the source table(epcgSourceTable)
7485     that corresponds to the current source"
7486     ::= { epcgReaderDeviceInformation 6 }
7487
7488 epcgRdrDevReboot    OBJECT-TYPE
7489 SYNTAX              TruthValue
7490 MAX-ACCESS          read-write
7491 STATUS             current
7492 DESCRIPTION
7493     "Setting this object to true(1) causes the device to reboot.
7494     Reading this object always returns false(2). Care should be taken
7495     that the reboot is done after the response is sent, to prevent the
7496     management station from resending the request and rebooting the device
7497     again."
7498     ::= { epcgReaderDeviceInformation 7 }
7499
7500 epcgRdrDevResetStatistics OBJECT-TYPE
7501 SYNTAX              TruthValue
7502 MAX-ACCESS          read-write
7503 STATUS             current
7504 DESCRIPTION
7505     " Object used to clear all measurements as follows.
7506     'true' - Causes the device to set all objects defined
7507             with a syntax of EpcgCurrentCountSinceReset
7508             to zero.
7509
7510     'false' - read value.
7511
7512     Reading this object always returns 'false'."
7513     ::= { epcgReaderDeviceInformation 8 }
7514
7515 epcgRdrDevResetTimestamp OBJECT-TYPE
7516 SYNTAX              DateAndTime
7517 MAX-ACCESS          read-only
7518 STATUS             current

```

```

7519 DESCRIPTION
7520 "This object is set to the current date and time when the
7521 epcgRdrDevResetStatistics object is set to 'true'.
7522 Initially it will be set to null value."
7523 ::= { epcgReaderDeviceInformation 9 }
7524
7525 epcgRdrDevNormalizePowerLevel OBJECT-TYPE
7526 SYNTAX TruthValue
7527 MAX-ACCESS read-only
7528 STATUS current
7529 DESCRIPTION
7530 " This object is used to indicate how an implementation
7531 presents power levels as follows.
7532
7533 'true' - All power levels will be converted be normalize
7534 to a range from 0-255.
7535
7536 'false' - All power levels are presented in raw format
7537 specific to a particular vendor.
7538
7539 This controls the presentation of the following objects.
7540 -- epcgAntRdPntPowerLevel
7541 "
7542 ::= { epcgReaderDeviceInformation 10 }
7543
7544 epcgRdrDevNormalizeNoiseLevel OBJECT-TYPE
7545 SYNTAX TruthValue
7546 MAX-ACCESS read-only
7547 STATUS current
7548 DESCRIPTION
7549 "This object is used to indicate how an implementation
7550 presents noise levels as follows.
7551
7552 'true' - All noise levels will be converted be normalize
7553 to a range from 0-255.
7554
7555 'false' - All noise levels are presented in raw format
7556 specific to a particular vendor.
7557
7558 This controls the presentation of the following objects.
7559 -- epcgAntRdPntNoiseLevel
7560 "
7561 ::= { epcgReaderDeviceInformation 11 }
7562
7563
7564 --*****
7565 --** Optional system wide counters
7566 --*****
7567 epcgGlobalCountersTable OBJECT-TYPE
7568 SYNTAX SEQUENCE OF EpcgGlobalCountersEntry
7569 MAX-ACCESS not-accessible
7570 STATUS current
7571 DESCRIPTION
7572 "The following table aggregates the various counters
7573 that are maintained per entity like read point,
7574 antenna read points, and source."
7575 ::= { epcgReaderDevice 2 }
7576
7577 epcgGlobalCountersEntry OBJECT-TYPE
7578 SYNTAX EpcgGlobalCountersEntry
7579 MAX-ACCESS not-accessible
7580 STATUS current

```

```

7581 DESCRIPTION
7582     "Each entry contains information need to communicate with
7583     each type of server. "
7584 INDEX { epcgGlobalCountersIndex }
7585 ::= { epcgGlobalCountersTable 1 }
7586
7587 EpcgGlobalCountersEntry ::= SEQUENCE {
7588     epcgGlobalCountersIndex INTEGER,
7589     epcgGlobalCountersData EpcgCurrentCountSinceReset
7590 }
7591
7592 epcgGlobalCountersIndex OBJECT-TYPE
7593     SYNTAX INTEGER { antennaTagsIdentified (1),
7594     antennaTagsNotIdentified (2),
7595     antennaMemoryReadOperations (18),
7596     antennaMemoryReadFailures (3),
7597     antennaWriteOperations (4),
7598     antennaWriteFailures (5),
7599     antennaKillOperations (6),
7600     antennaKillFailures (7),
7601     antennaEraseOperations (8),
7602     antennaEraseFailures (19),
7603     antennaLockOperations (9),
7604     antennaLockFailures (10),
7605     sourceUnknownToGlimpsed (11),
7606     sourceGlimpsedToUnknown (12),
7607     sourceGlimpsedToObserved (13),
7608     sourceObservedToLost (14),
7609     sourceLostToGlimpsed (15),
7610     sourceLostToUnknown (16),
7611     triggerMatches (17)
7612 }
7613 MAX-ACCESS not-accessible
7614 STATUS current
7615 DESCRIPTION
7616     "Index to access global counters as follows. The units of
7617     each counter will be the same as object indicated by
7618     index. For example the antennaIdentifier value references
7619     the epcgAntRdPntTagsIdentified and it uses 'tags' as units.
7620
7621     antennaTagsIdentified - Aggregate counter for all instances
7622     of the epcgAntRdPntTagsIdentified object.
7623
7624     antennaTagsNotIdentified - Aggregate counter for all
7625     instances of the epcgAntRdPntTagsNotIdentified object.
7626
7627     antennaMemoryReadOperations - Aggregate counter for all instances
7628     of the epcgAntRdPntMemoryReadOperations object.
7629
7630     antennaMemoryReadFailures - Aggregate counter for all instances
7631     of the epcgAntRdPntMemoryReadFailures object.
7632
7633     antennaWriteOperations - Aggregate counter for all instances
7634     of the epcgAntRdPntWriteOperations object.
7635
7636     antennaWriteFailures - Aggregate counter for all instances
7637     of the epcgAntRdPntWriteFailures object.
7638
7639     antennaKillOperations - Aggregate counter for all instances
7640     of the epcgAntRdPntKillOperations object.
7641
7642     antennaKillFailures - Aggregate counter for all instances

```



```

7643         of the epcgAntRdPntKillFailures object.
7644
7645     antennaEraseOperations - Aggregate counter for all instances
7646         of the epcgAntRdPntEraseOperations object.
7647
7648     antennaEraseFailures - Aggregate counter for all instances
7649         of the epcgAntRdPntEraseFailures object.
7650
7651     antennaLockOperations - Aggregate counter for all instances
7652         of the epcgAntRdPntLockOperations object.
7653
7654     antennaLockFailures - Aggregate counter for all instances
7655         of the epcgAntRdPntLockFailures object.
7656
7657     sourceUnknownToGlimpsed - Aggregate counter for all instances
7658         of the epcgSrcUnknownToGlimpsedTrans object.
7659
7660     sourceGlimpsedToUnknown - Aggregate counter for all instances
7661         of the epcgSrcGlimpsedToUnknownTrans object.
7662
7663     sourceGlimpsedToObserved - Aggregate counter for all instances
7664         of the epcgSrcGlimpsedToObservedTrans object.
7665
7666     sourceObservedToLost - Aggregate counter for all instances
7667         of the epcgSrcObservedToLostTrans object.
7668
7669     sourceLostToGlimpsed - Aggregate counter for all instances
7670         of the epcgSrcLostToGlimpsedTrans object.
7671
7672     sourceLostToUnknown - Aggregate counter for all instances
7673         of the epcgSrcLostToUnknownTrans object.
7674
7675     triggerMatches - Aggregate counter for all instances
7676         of the epcgTriggerMatches object.
7677
7678     source - Aggregate counter for all instances
7679         of the object.
7680     "
7681     ::= { epcgGlobalCountersEntry 1 }
7682
7683 epcgGlobalCountersData OBJECT-TYPE
7684     SYNTAX      EpcgCurrentCountSinceReset
7685     MAX-ACCESS  read-only
7686     STATUS      current
7687     DESCRIPTION
7688         "The object contain aggregate count for all instances
7689         of object as defined by epcgGlobalCountersIndex."
7690     ::= { epcgGlobalCountersEntry 2 }
7691
7692 --*****
7693 --** Reader Device - Operation Objects
7694 --*****
7695 epcgReaderDeviceOperation
7696     OBJECT IDENTIFIER ::= { epcgReaderDevice 3 }
7697
7698 epcgRdrDevOperStatus OBJECT-TYPE
7699     SYNTAX      EpcgOperationalStatus
7700     MAX-ACCESS  read-only
7701     STATUS      current
7702     DESCRIPTION
7703         "Current status of the reader device."
7704     ::= { epcgReaderDeviceOperation 1}

```

```

7705
7706 epcgRdrDevOperStatusPrior OBJECT-TYPE
7707     SYNTAX          EpcgOperationalStatus
7708     MAX-ACCESS     accessible-for-notify
7709     STATUS          current
7710     DESCRIPTION
7711         "Status of reader prior to status change that triggered
7712         the epcgReaderDeviceOperationState notification."
7713     ::= { epcgReaderDeviceOperation 2 }
7714
7715 epcgRdrDevOperStateEnable OBJECT-TYPE
7716     SYNTAX          TruthValue
7717     MAX-ACCESS     read-write
7718     STATUS          current
7719     DESCRIPTION
7720         "This object controls the generation of the
7721         epcgReaderDeviceOperationState notification
7722         as follows.
7723
7724         'true' Indicates that the notification of
7725         operation state changes has been
7726         enabled.
7727
7728         'false' Indicates that the notification of
7729         operation state changes has been
7730         disabled."
7731     DEFVAL { false }
7732     ::= { epcgReaderDeviceOperation 3 }
7733
7734 epcgRdrDevOperNotifFromState OBJECT-TYPE
7735     SYNTAX          EpcgOperationalEnable
7736     MAX-ACCESS     read-write
7737     STATUS          current
7738     DESCRIPTION
7739         "The source state used to control the generation of
7740         the epcgReaderDeviceOperationState notification."
7741     DEFVAL { { unknown, other, up, down } }
7742     ::= { epcgReaderDeviceOperation 4 }
7743
7744 epcgRdrDevOperNotifToState OBJECT-TYPE
7745     SYNTAX          EpcgOperationalEnable
7746     MAX-ACCESS     read-write
7747     STATUS          current
7748     DESCRIPTION
7749         "The destination state used to control the generation of
7750         the epcgReaderDeviceOperationState notification."
7751     DEFVAL { { unknown, other, up, down } }
7752     ::= { epcgReaderDeviceOperation 5 }
7753
7754 epcgRdrDevOperNotifStateLevel OBJECT-TYPE
7755     SYNTAX          EpcgNotifLevel
7756     MAX-ACCESS     read-write
7757     STATUS          current
7758     DESCRIPTION
7759         "The severity level assigned to the
7760         epcgReaderDeviceOperationStatenotification"
7761     DEFVAL { error }
7762     ::= { epcgReaderDeviceOperation 6 }
7763
7764 epcgRdrDevOperStateSuppressInterval OBJECT-TYPE
7765     SYNTAX          Unsigned32 (0 | 1..3600 )
7766     UNITS           "seconds"

```

```

7767     MAX-ACCESS    read-write
7768     STATUS        current
7769     DESCRIPTION
7770         "The length of the interval used to suppress generation
7771         of epcgReaderDeviceOperationState change notifications.
7772
7773         Notification will not be suppressed when zero value is
7774         specified."
7775     DEFVAL { 0 }
7776     ::= { epcgReaderDeviceOperation 7 }
7777
7778 epcgRdrDevOperStateSuppressions OBJECT-TYPE
7779     SYNTAX          Counter32
7780     UNITS           "notifications"
7781     MAX-ACCESS      read-only
7782     STATUS          current
7783     DESCRIPTION
7784         "The number of device operation state change
7785         notifications that have been suppressed."
7786     ::= { epcgReaderDeviceOperation 8 }
7787
7788 --*****
7789 --** Reader Device Memory Objects **
7790 --*****
7791 epcgReaderDeviceMemory
7792     OBJECT IDENTIFIER ::= { epcgReaderDevice 4 }
7793
7794 epcgRdrDevFreeMemory      OBJECT-TYPE
7795     SYNTAX                  Gauge32(0..4294967295)
7796     UNITS                    "bytes"
7797     MAX-ACCESS              read-only
7798     STATUS                  current
7799     DESCRIPTION
7800         "Reader device free memory, in bytes"
7801     ::= { epcgReaderDeviceMemory 1 }
7802
7803 epcgRdrDevFreeMemoryNotifEnable OBJECT-TYPE
7804     SYNTAX                  TruthValue
7805     MAX-ACCESS              read-write
7806     STATUS                  current
7807     DESCRIPTION
7808         "This object controls the generation of the
7809         epcgRdrDevMemoryState notification as follows.
7810
7811         'true'  Indicates that the notification of
7812         epcgRdrDevMemoryState changes has been enabled.
7813
7814         'false' Indicates that the notification of
7815         epcgRdrDevMemoryState changes has been disabled."
7816     ::= { epcgReaderDeviceMemory 2 }
7817
7818 epcgRdrDevFreeMemoryNotifLevel OBJECT-TYPE
7819     SYNTAX                  EpcgNotifLevel
7820     MAX-ACCESS              read-write
7821     STATUS                  current
7822     DESCRIPTION
7823         "The severity level assigned to the epcgRdrDevMemoryState
7824         notification"
7825     DEFVAL { critical }
7826     ::= { epcgReaderDeviceMemory 3 }
7827
7828 epcgRdrDevFreeMemoryOnsetThreshold OBJECT-TYPE

```

```

7829     SYNTAX                EpcgThreshold
7830     MAX-ACCESS             read-write
7831     STATUS                 current
7832     DESCRIPTION
7833         "The threshold value used to detect low memory situations."
7834     ::= { epcgReaderDeviceMemory 4 }
7835
7836     epcgRdrDevFreeMemoryAbateThreshold OBJECT-TYPE
7837     SYNTAX                EpcgThreshold
7838     MAX-ACCESS             read-write
7839     STATUS                 current
7840     DESCRIPTION
7841         "The threshold value used to detect end of low memory
7842         situations."
7843     ::= { epcgReaderDeviceMemory 5 }
7844
7845     epcgRdrDevFreeMemoryStatus OBJECT-TYPE
7846     SYNTAX                INTEGER { shortage (1),
7847                             normal (2)
7848                             }
7849     MAX-ACCESS             read-only
7850     STATUS                 current
7851     DESCRIPTION
7852         "The current memory state."
7853     ::= { epcgReaderDeviceMemory 6 }
7854
7855     epcgRdrDevMemStateSuppressInterval OBJECT-TYPE
7856     SYNTAX                Unsigned32 (0 | 1..3600 )
7857     UNITS                  "seconds"
7858     MAX-ACCESS             read-write
7859     STATUS                 current
7860     DESCRIPTION
7861         "The length of the interval used to suppress generation
7862         of epcgRdrDevMemoryState change notifications.
7863
7864         Notification will not be suppressed when zero value is
7865         specified."
7866     DEFVAL { 0 }
7867     ::= { epcgReaderDeviceMemory 7 }
7868
7869     epcgRdrDevMemStateSuppressions OBJECT-TYPE
7870     SYNTAX                Counter32
7871     UNITS                  "notifications"
7872     MAX-ACCESS             read-only
7873     STATUS                 current
7874     DESCRIPTION
7875         "The number of device memory state change
7876         notifications that have been suppressed."
7877     ::= { epcgReaderDeviceMemory 8 }
7878
7879
7880     --*****
7881     --** Reader Device - Server Table
7882     --*****
7883     epcgReaderServerTable OBJECT-TYPE
7884     SYNTAX                SEQUENCE OF EpcgReaderServerEntry
7885     MAX-ACCESS             not-accessible
7886     STATUS                 current
7887     DESCRIPTION
7888         "The following table provide information about the various
7889         servers used by reader device.  Entries are added and removed
7890         from this table via epcgReaderServerRowStatus in accordance

```

```

7891         with the RowStatus convention."
7892 ::= { epcgReaderDevice 5 }
7893
7894 epcgReaderServerEntry OBJECT-TYPE
7895     SYNTAX          EpcgReaderServerEntry
7896     MAX-ACCESS     not-accessible
7897     STATUS         current
7898     DESCRIPTION
7899         "Each entry contains information need to communicate with
7900         each type of server. "
7901     INDEX { epcgReaderServerType,
7902            epcgReaderServerNumber
7903           }
7904 ::= { epcgReaderServerTable 1 }
7905
7906 EpcgReaderServerEntry ::= SEQUENCE {
7907     epcgReaderServerType          INTEGER,
7908     epcgReaderServerNumber        Unsigned32,
7909     epcgReaderServerAddressType   InetAddressType,
7910     epcgReaderServerAddress       InetAddress,
7911     epcgReaderServerRowStatus     RowStatus
7912 }
7913
7914 epcgReaderServerType          OBJECT-TYPE
7915     SYNTAX          INTEGER { dhcp (1),
7916                          dns (2),
7917                          time (3)
7918                       }
7919     MAX-ACCESS     not-accessible
7920     STATUS         current
7921     DESCRIPTION
7922         "The type of server as follows.
7923
7924         dhcp - DHCP server.
7925         dns  - Domain Name System.
7926         time - Time server used to synchronization Reader's
7927               real-time clock.
7928         "
7929 ::= { epcgReaderServerEntry 1 }
7930
7931 epcgReaderServerNumber        OBJECT-TYPE
7932     SYNTAX          Unsigned32(1..10)
7933     MAX-ACCESS     not-accessible
7934     STATUS         current
7935     DESCRIPTION
7936         "The server number. This index is assigned arbitrarily by
7937         the SNMP engine and is not required to be saved over
7938         restarts."
7939 ::= { epcgReaderServerEntry 2 }
7940
7941 epcgReaderServerAddressType   OBJECT-TYPE
7942     SYNTAX          InetAddressType
7943     MAX-ACCESS     read-create
7944     STATUS         current
7945     DESCRIPTION
7946         "The type of Internet address to be used to
7947         communicate with server to collect."
7948 ::= { epcgReaderServerEntry 3 }
7949
7950 epcgReaderServerAddress       OBJECT-TYPE
7951     SYNTAX          InetAddress
7952     MAX-ACCESS     read-create

```

```

7953 STATUS current
7954 DESCRIPTION
7955 "The address of the server and the format of the address
7956 is specified by the epcgReaderServerAddressType object."
7957 ::= { epcgReaderServerEntry 4 }
7958
7959 epcgReaderServerRowStatus OBJECT-TYPE
7960 SYNTAX RowStatus
7961 MAX-ACCESS read-create
7962 STATUS current
7963 DESCRIPTION
7964 "The status of this row."
7965 ::= { epcgReaderServerEntry 5 }
7966
7967
7968 --*****
7969 --** Read Points **
7970 --*****
7971 epcgReadPointTable OBJECT-TYPE
7972 SYNTAX SEQUENCE OF EpcgReadPointEntry
7973 MAX-ACCESS not-accessible
7974 STATUS current
7975 DESCRIPTION
7976 "This table contains a row entry for each of the Reader's
7977 ReadPoints. A ReadPoint can be any physical entity that is
7978 capable of acquiring (or distributing, in the case of tag
7979 programming) item data. A single RF tag reader antenna is
7980 one example of a ReadPoint. ReadPoints are not limited to
7981 antennas; for example, a bar code scanning device is another
7982 example of a ReadPoint. Hence, ReadPoint is the base class
7983 for all physical elements over which EPC data is read or
7984 written.
7985
7986 The AntennaReadPoint class extends this class and is
7987 represented by the AntennaReadPointTable. (A later revision
7988 of this MIB may introduce a BarCodeReadPointTable that
7989 augments this table.)
7990
7991 Typically the number of rows in this table will be equal to
7992 the number of Antenna ports on a reader"
7993 ::= { epcgReadPoints 1 }
7994
7995
7996 epcgReadPointEntry OBJECT-TYPE
7997 SYNTAX EpcgReadPointEntry
7998 MAX-ACCESS not-accessible
7999 STATUS current
8000 DESCRIPTION
8001 "The properties of a read point. SNMP operations can
8002 neither create nor delete rows of this table"
8003 INDEX { epcgReadPointIndex }
8004 ::= { epcgReadPointTable 1 }
8005
8006 EpcgReadPointEntry ::= SEQUENCE {
8007 epcgReadPointIndex Unsigned32,
8008 epcgReadPointName SnmpAdminString,
8009 epcgReadPointDescription SnmpAdminString,
8010 epcgReadPointAdminStatus EpcgAdministrativeStatus,
8011 epcgReadPointOperStatus EpcgOperationalStatus,
8012 epcgReadPointOperStateNotifyEnable TruthValue,
8013 epcgReadPointOperNotifyFromState EpcgOperationalEnable,
8014 epcgReadPointOperNotifyToState EpcgOperationalEnable,

```

```

8015         epcgReadPointOperNotifyStateLevel      EpcgNotifLevel,
8016         epcgReadPointOperStatusPrior           EpcgOperationalStatus,
8017         epcgReadPointOperStateSuppressInterval Unsigned32,
8018         epcgReadPointOperStateSuppressions     Counter32
8019     }
8020
8021 epcgReadPointIndex OBJECT-TYPE
8022     SYNTAX      Unsigned32(1..2147483647)
8023     MAX-ACCESS  not-accessible
8024     STATUS      current
8025     DESCRIPTION
8026         "Index used for uniquely identifying a read point within
8027         the scope of a given RFID Reader. This index is assigned
8028         arbitrarily by the SNMP engine and is not required to be
8029         saved over restarts."
8030     ::= { epcgReadPointEntry 1 }
8031
8032 epcgReadPointName      OBJECT-TYPE
8033     SYNTAX      SnmpAdminString
8034     MAX-ACCESS  read-only
8035     STATUS      current
8036     DESCRIPTION
8037         "Unique name assigned to read point by device."
8038     ::= { epcgReadPointEntry 2 }
8039
8040 epcgReadPointDescription OBJECT-TYPE
8041     SYNTAX      SnmpAdminString
8042     MAX-ACCESS  read-only
8043     STATUS      current
8044     DESCRIPTION
8045         "A textual description of the read point."
8046     ::= { epcgReadPointEntry 3 }
8047
8048 epcgReadPointAdminStatus OBJECT-TYPE
8049     SYNTAX      EpcgAdministrativeStatus
8050     MAX-ACCESS  read-write
8051     STATUS      current
8052     DESCRIPTION
8053         "The administratively specified status of the read point."
8054     ::= { epcgReadPointEntry 4 }
8055
8056 epcgReadPointOperStatus OBJECT-TYPE
8057     SYNTAX      EpcgOperationalStatus
8058     MAX-ACCESS  read-only
8059     STATUS      current
8060     DESCRIPTION
8061         "The read point's current operational status"
8062     ::= { epcgReadPointEntry 5 }
8063
8064 epcgReadPointOperStateNotifyEnable OBJECT-TYPE
8065     SYNTAX      TruthValue
8066     MAX-ACCESS  read-write
8067     STATUS      current
8068     DESCRIPTION
8069         "This object controls the generation of the
8070         epcgReadPointOperationState notification
8071         for this read point as follows.
8072
8073         'true' Indicates that the notification of read point
8074         operation state changes has been enabled.
8075
8076         'false' Indicates that the notification of read point

```

```

8077         operation state changes has been disable."
8078
8079     ::= { epcgReadPointEntry 6 }
8080
8081 epcgReadPointOperNotifyFromState OBJECT-TYPE
8082     SYNTAX          EpcgOperationalEnable
8083     MAX-ACCESS      read-write
8084     STATUS          current
8085     DESCRIPTION
8086         "The source state used to control the generation of
8087         the epcgReadPointOperationState notification."
8088     DEFVAL { { unknown, other, up, down} }
8089     ::= { epcgReadPointEntry 7 }
8090
8091 epcgReadPointOperNotifyToState OBJECT-TYPE
8092     SYNTAX          EpcgOperationalEnable
8093     MAX-ACCESS      read-write
8094     STATUS          current
8095     DESCRIPTION
8096         "The destination state used to control the generation of
8097         the epcgReadPointOperationState notification."
8098     DEFVAL { { unknown, other, up, down} }
8099     ::= { epcgReadPointEntry 8 }
8100
8101 epcgReadPointOperNotifyStateLevel OBJECT-TYPE
8102     SYNTAX          EpcgNotifLevel
8103     MAX-ACCESS      read-write
8104     STATUS          current
8105     DESCRIPTION
8106         "The severity level assigned to the
8107         epcgReadPointOperationState notification."
8108     DEFVAL { error }
8109     ::= { epcgReadPointEntry 9 }
8110
8111 epcgReadPointOperStatusPrior OBJECT-TYPE
8112     SYNTAX          EpcgOperationalStatus
8113     MAX-ACCESS      accessible-for-notify
8114     STATUS          current
8115     DESCRIPTION
8116         "The past status of read-point prior to the new status change
8117         that triggered the epcgReadPointOperationState
8118         notificatation."
8119     ::= { epcgReadPointEntry 10 }
8120
8121 epcgReadPointOperStateSupressInterval OBJECT-TYPE
8122     SYNTAX          Unsigned32 (0 | 1..3600 )
8123     UNITS           "seconds"
8124     MAX-ACCESS      read-write
8125     STATUS          current
8126     DESCRIPTION
8127         "The length of the interval used to suppress generation
8128         of epcgReadPointOperationState change notifications.
8129
8130         Notification will not be suppressed when zero value is
8131         specified."
8132     DEFVAL { 0 }
8133     ::= { epcgReadPointEntry 11 }
8134
8135 epcgReadPointOperStateSuppressions OBJECT-TYPE
8136     SYNTAX          Counter32
8137     UNITS           "notifications"
8138     MAX-ACCESS      read-only

```



```

8139     STATUS          current
8140     DESCRIPTION
8141         "The number of read point operational state change
8142         notifications that have been suppressed."
8143     ::= { epcgReadPointEntry 12 }
8144
8145
8146 --*****
8147 --** Antenna Read Points **
8148 --*****
8149 epcgAntReadPointTable OBJECT-TYPE
8150     SYNTAX          SEQUENCE OF EpcgAntReadPointEntry
8151     MAX-ACCESS     not-accessible
8152     STATUS          current
8153     DESCRIPTION
8154         "This table extends the epcgReadPointTable. It contains
8155         one row for each physical antenna, or antenna port, on
8156         a Reader."
8157     ::= { epcgAntennaReadPoints 1 }
8158
8159 epcgAntReadPointEntry OBJECT-TYPE
8160     SYNTAX          EpcgAntReadPointEntry
8161     MAX-ACCESS     not-accessible
8162     STATUS          current
8163     DESCRIPTION
8164         "The properties of an antenna read point.  SNMP
8165         operations can neither create nor delete
8166         rows of this table"
8167
8168     INDEX { epcgReadPointIndex }
8169     ::= { epcgAntReadPointTable 1 }
8170
8171 EpcgAntReadPointEntry ::= SEQUENCE {
8172     epcgAntRdPntTagsIdentified          EpcgCurrentCountSinceReset,
8173     epcgAntRdPntTagsNotIdentified       EpcgCurrentCountSinceReset,
8174     epcgAntRdPntMemoryReadOperations    EpcgCurrentCountSinceReset,
8175     epcgAntRdPntMemoryReadFailures      EpcgCurrentCountSinceReset,
8176     epcgAntRdPntReadFailureNotifEnable  TruthValue,
8177     epcgAntRdPntReadFailureNotifLevel   EpcgNotifLevel,
8178     epcgAntRdPntReadFailureSuppressInterval Unsigned32,
8179     epcgAntRdPntReadFailureSuppressions Counter32,
8180     epcgAntRdPntWriteOperations          EpcgCurrentCountSinceReset,
8181     epcgAntRdPntWriteFailures            EpcgCurrentCountSinceReset,
8182     epcgAntRdPntWriteFailuresNotifEnable TruthValue,
8183     epcgAntRdPntWriteFailuresNotifLevel EpcgNotifLevel,
8184     epcgAntRdPntWriteFailureSuppressInterval Unsigned32,
8185     epcgAntRdPntWriteFailureSuppressions Counter32,
8186     epcgAntRdPntKillOperations           EpcgCurrentCountSinceReset,
8187     epcgAntRdPntKillFailures             EpcgCurrentCountSinceReset,
8188     epcgAntRdPntKillFailuresNotifEnable  TruthValue,
8189     epcgAntRdPntKillFailuresNotifLevel   EpcgNotifLevel,
8190     epcgAntRdPntKillFailureSuppressInterval Unsigned32,
8191     epcgAntRdPntKillFailureSuppressions Counter32,
8192     epcgAntRdPntEraseOperations          EpcgCurrentCountSinceReset,
8193     epcgAntRdPntEraseFailures            EpcgCurrentCountSinceReset,
8194     epcgAntRdPntEraseFailuresNotifEnable TruthValue,
8195     epcgAntRdPntEraseFailuresNotifLevel EpcgNotifLevel,
8196     epcgAntRdPntEraseFailureSuppressInterval Unsigned32,
8197     epcgAntRdPntEraseFailureSuppressions Counter32,
8198     epcgAntRdPntLockOperations            EpcgCurrentCountSinceReset,
8199     epcgAntRdPntLockFailures             EpcgCurrentCountSinceReset,
8200     epcgAntRdPntLockFailuresNotifEnable  TruthValue,

```

```

8201     epcgAntRdPntLockFailuresNotifLevel  EpcgNotifLevel,
8202     epcgAntRdPntLockFailureSuppressInterval  Unsigned32,
8203     epcgAntRdPntLockFailureSuppressions  Counter32,
8204     epcgAntRdPntPowerLevel  Integer32,
8205     epcgAntRdPntNoiseLevel  Integer32,
8206     epcgAntRdPntTimeEnergized  EpcgCurrentCountSinceReset
8207 }
8208
8209 epcgAntRdPntTagsIdentified OBJECT-TYPE
8210     SYNTAX      EpcgCurrentCountSinceReset
8211     UNITS       "tags"
8212     MAX-ACCESS  read-only
8213     STATUS      current
8214     DESCRIPTION
8215         "Count of tags successfully identified by this antenna read point."
8216     ::= { epcgAntReadPointEntry 1 }
8217
8218 epcgAntRdPntTagsNotIdentified OBJECT-TYPE
8219     SYNTAX      EpcgCurrentCountSinceReset
8220     UNITS       "tags"
8221     MAX-ACCESS  read-only
8222     STATUS      current
8223     DESCRIPTION
8224         "Count of tags failed to be identified by this antenna
8225         read point.  If the device does not implement the technology to
8226         unambiguously determine a failed tag singulation, it is acceptable to
8227         leave this value at zero."
8228     ::= { epcgAntReadPointEntry 2 }
8229
8230 epcgAntRdPntMemoryReadOperations OBJECT-TYPE
8231     SYNTAX      EpcgCurrentCountSinceReset
8232     UNITS       "tags"
8233     MAX-ACCESS  read-only
8234     STATUS      current
8235     DESCRIPTION
8236         "Count of tags successfully having their memory read by this
8237         antenna read point."
8238     ::= { epcgAntReadPointEntry 25 }
8239
8240 epcgAntRdPntMemoryReadFailures OBJECT-TYPE
8241     SYNTAX      EpcgCurrentCountSinceReset
8242     UNITS       "tags"
8243     MAX-ACCESS  read-only
8244     STATUS      current
8245     DESCRIPTION
8246         "Count of tags failed to have their memory read by this
8247         antenna read point."
8248     ::= { epcgAntReadPointEntry 3 }
8249
8250 epcgAntRdPntReadFailureNotifEnable OBJECT-TYPE
8251     SYNTAX      TruthValue
8252     MAX-ACCESS  read-write
8253     STATUS      current
8254     DESCRIPTION
8255         "This object controls the generation of the
8256         epcgReaderAntennaReadFailure notification as follows.
8257
8258         'true' Indicates that the notification of antenna read
8259         failure has been enabled.
8260
8261         'false' Indicates that the notification of antenna read
8262         failure changes has been disabled."

```

```

8263 ::= { epcgAntReadPointEntry 4 }
8264
8265 epcgAntRdPntReadFailureNotifLevel OBJECT-TYPE
8266 SYNTAX      EpcgNotifLevel
8267 MAX-ACCESS  read-write
8268 STATUS      current
8269 DESCRIPTION
8270     "The severity level assigned to the
8271     epcgReaderAntennaReadFailure notification."
8272 DEFVAL { error }
8273 ::= { epcgAntReadPointEntry 5 }
8274
8275 epcgAntRdPntReadFailureSupressInterval OBJECT-TYPE
8276 SYNTAX      Unsigned32 (0 | 1..3600 )
8277 UNITS       "seconds"
8278 MAX-ACCESS  read-write
8279 STATUS      current
8280 DESCRIPTION
8281     "The length of the interval used to suppress generation
8282     of epcgReaderAntennaReadFailure notifications.
8283
8284     Notification will not be suppressed when zero value is
8285     specified."
8286 DEFVAL { 0 }
8287 ::= { epcgAntReadPointEntry 26 }
8288
8289 epcgAntRdPntReadFailureSuppressions OBJECT-TYPE
8290 SYNTAX      Counter32
8291 UNITS       "notifications"
8292 MAX-ACCESS  read-only
8293 STATUS      current
8294 DESCRIPTION
8295     "The number of epcgReaderAntennaReadFailure
8296     notifications that have been suppressed."
8297 ::= { epcgAntReadPointEntry 27 }
8298
8299 epcgAntRdPntWriteOperations OBJECT-TYPE
8300 SYNTAX      EpcgCurrentCountSinceReset
8301 UNITS       "write operations"
8302 MAX-ACCESS  read-only
8303 STATUS      current
8304 DESCRIPTION
8305     "Number of writes successfully done by this antenna read point."
8306 ::= { epcgAntReadPointEntry 6 }
8307
8308 epcgAntRdPntWriteFailures OBJECT-TYPE
8309 SYNTAX      EpcgCurrentCountSinceReset
8310 UNITS       "write operations"
8311 MAX-ACCESS  read-only
8312 STATUS      current
8313 DESCRIPTION
8314     "Number of writes attempted and failed by this antenna
8315     read point."
8316 ::= { epcgAntReadPointEntry 7 }
8317
8318 epcgAntRdPntWriteFailuresNotifEnable OBJECT-TYPE
8319 SYNTAX      TruthValue
8320 MAX-ACCESS  read-write
8321 STATUS      current
8322 DESCRIPTION
8323     "This object controls the generation of the
8324     epcgReaderAntennaWriteFailure notification as follows.

```

```

8325
8326     'true' Indicates that the notification of antenna write
8327         failure changes has been enabled.
8328
8329     'false' Indicates that the notification of ntenna write
8330         failure changes has been disabled."
8331 ::= { epcgAntReadPointEntry 8 }
8332
8333 epcgAntRdPntWriteFailuresNotifLevel OBJECT-TYPE
8334     SYNTAX          EpcgNotifLevel
8335     MAX-ACCESS      read-write
8336     STATUS          current
8337     DESCRIPTION
8338         "The severity level assigned to the
8339         epcgReaderAntennaWriteFailure notification."
8340     DEFVAL { notice }
8341     ::= { epcgAntReadPointEntry 9 }
8342
8343 epcgAntRdPntWriteFailureSupprssInterval OBJECT-TYPE
8344     SYNTAX          Unsigned32 (0 | 1..3600 )
8345     UNITS           "seconds"
8346     MAX-ACCESS      read-write
8347     STATUS          current
8348     DESCRIPTION
8349         "The length of the interval used to suppress generation
8350         of epcgReaderAntennaWriteFailure notifications.
8351
8352         Notification will not be suppressed when zero value is
8353         specified."
8354     DEFVAL { 0 }
8355     ::= { epcgAntReadPointEntry 28 }
8356
8357 epcgAntRdPntWriteFailureSuppressions OBJECT-TYPE
8358     SYNTAX          Counter32
8359     UNITS           "notifications"
8360     MAX-ACCESS      read-only
8361     STATUS          current
8362     DESCRIPTION
8363         "The number of epcgReaderAntennaWriteFailure
8364         notifications that have been suppressed."
8365     ::= { epcgAntReadPointEntry 29 }
8366
8367 epcgAntRdPntKillOperations OBJECT-TYPE
8368     SYNTAX          EpcgCurrentCountSinceReset
8369     UNITS           "kill operations"
8370     MAX-ACCESS      read-only
8371     STATUS          current
8372     DESCRIPTION
8373         "Number of kill operations successfully done by this antenna read
8374         point."
8375     ::= { epcgAntReadPointEntry 10 }
8376
8377 epcgAntRdPntKillFailures OBJECT-TYPE
8378     SYNTAX          EpcgCurrentCountSinceReset
8379     UNITS           "kill operations"
8380     MAX-ACCESS      read-only
8381     STATUS          current
8382     DESCRIPTION
8383         "Number of kill operations attempted and failed by this
8384         antenna read point."
8385     ::= { epcgAntReadPointEntry 11 }
8386

```

```

8387 epcgAntRdPntKillFailuresNotifEnable OBJECT-TYPE
8388     SYNTAX          TruthValue
8389     MAX-ACCESS      read-write
8390     STATUS          current
8391     DESCRIPTION
8392         "This object controls the generation of the
8393         epcgReaderAntennaKillFailure notification as follows.
8394
8395         'true' Indicates that the notification of antenna kill
8396         failure has been enabled.
8397
8398         'false' Indicates that the notification of antenna kill
8399         failure has been disabled."
8400     ::= { epcgAntReadPointEntry 12 }
8401
8402 epcgAntRdPntKillFailuresNotifLevel OBJECT-TYPE
8403     SYNTAX          EpcgNotifLevel
8404     MAX-ACCESS      read-write
8405     STATUS          current
8406     DESCRIPTION
8407         "The severity level assigned to the
8408         epcgReaderAntennaKillFailure notification."
8409     DEFVAL { notice }
8410     ::= { epcgAntReadPointEntry 13 }
8411
8412 epcgAntRdPntKillFailureSuppressInterval OBJECT-TYPE
8413     SYNTAX          Unsigned32 (0 | 1..3600 )
8414     UNITS           "seconds"
8415     MAX-ACCESS      read-write
8416     STATUS          current
8417     DESCRIPTION
8418         "The length of the interval used to suppress generation
8419         of epcgReaderAntennaKillFailure notifications.
8420
8421         Notification will not be suppressed when zero value is
8422         specified."
8423     DEFVAL { 0 }
8424     ::= { epcgAntReadPointEntry 30 }
8425
8426 epcgAntRdPntKillFailureSuppressions OBJECT-TYPE
8427     SYNTAX          Counter32
8428     UNITS           "notifications"
8429     MAX-ACCESS      read-only
8430     STATUS          current
8431     DESCRIPTION
8432         "The number of epcgReaderAntennaKillFailure
8433         notifications that have been suppressed."
8434     ::= { epcgAntReadPointEntry 31 }
8435
8436 epcgAntRdPntEraseOperations OBJECT-TYPE
8437     SYNTAX          EpcgCurrentCountSinceReset
8438     UNITS           "erase operations"
8439     MAX-ACCESS      read-only
8440     STATUS          current
8441     DESCRIPTION
8442         "Number of erase operations successfully completed by this antenna
8443         read point. "
8444     ::= { epcgAntReadPointEntry 14 }
8445
8446 epcgAntRdPntEraseFailures OBJECT-TYPE
8447     SYNTAX          EpcgCurrentCountSinceReset
8448     UNITS           "erase operations"

```

```

8449     MAX-ACCESS          read-only
8450     STATUS              current
8451     DESCRIPTION
8452         "Number of erase operations attempted and failed by this
8453         antenna read point."
8454     ::= { epcgAntReadPointEntry 15 }
8455
8456     epcgAntRdPntEraseFailuresNotifEnable OBJECT-TYPE
8457     SYNTAX              TruthValue
8458     MAX-ACCESS          read-write
8459     STATUS              current
8460     DESCRIPTION
8461         "This object controls the generation of the
8462         epcgReaderAntennaEraseFailure notification as follows.
8463
8464         'true' Indicates that the notification of antenna erase
8465         failure changes has been enabled.
8466
8467         'false' Indicates that the notification of antenna erase failure
8468         changes has been disabled."
8469     ::= { epcgAntReadPointEntry 16 }
8470
8471     epcgAntRdPntEraseFailuresNotifLevel OBJECT-TYPE
8472     SYNTAX              EpcgNotifLevel
8473     MAX-ACCESS          read-write
8474     STATUS              current
8475     DESCRIPTION
8476         "The severity level assigned to the
8477         epcgReaderAntennaEraseFailure notification."
8478     DEFVAL { notice }
8479     ::= { epcgAntReadPointEntry 17 }
8480
8481     epcgAntRdPntEraseFailureSuppressInterval OBJECT-TYPE
8482     SYNTAX              Unsigned32 ( 0 | 1..3600 )
8483     UNITS                "seconds"
8484     MAX-ACCESS          read-write
8485     STATUS              current
8486     DESCRIPTION
8487         "The length of the interval used to suppress generation
8488         of epcgReaderAntennaEraseFailure notifications.
8489
8490         Notification will not be suppressed when zero value is
8491         specified."
8492     DEFVAL { 0 }
8493     ::= { epcgAntReadPointEntry 32 }
8494
8495     epcgAntRdPntEraseFailureSuppressions OBJECT-TYPE
8496     SYNTAX              Counter32
8497     UNITS                "notifications"
8498     MAX-ACCESS          read-only
8499     STATUS              current
8500     DESCRIPTION
8501         "The number of epcgReaderAntennaEraseFailure
8502         notifications that have been suppressed."
8503     ::= { epcgAntReadPointEntry 33 }
8504
8505     epcgAntRdPntLockOperations OBJECT-TYPE
8506     SYNTAX              EpcgCurrentCountSinceReset
8507     UNITS                "lock operations"
8508     MAX-ACCESS          read-only
8509     STATUS              current
8510     DESCRIPTION

```

```

8511         "Number of lock operations successfully issued by this antenna
8512         read point."
8513         ::= { epcgAntReadPointEntry 18 }
8514
8515 epcgAntRdPntLockFailures OBJECT-TYPE
8516     SYNTAX      EpcgCurrentCountSinceReset
8517     UNITS       "lock operations"
8518     MAX-ACCESS  read-only
8519     STATUS      current
8520     DESCRIPTION
8521         "Number of lock operations attempted and failed by this
8522         antenna read point."
8523     ::= { epcgAntReadPointEntry 19 }
8524
8525 epcgAntRdPntLockFailuresNotifEnable OBJECT-TYPE
8526     SYNTAX      TruthValue
8527     MAX-ACCESS  read-write
8528     STATUS      current
8529     DESCRIPTION
8530         "This object controls the generation of the
8531         epcgReaderAntennaLockFailure notification
8532         as follows.
8533
8534         'true' Indicates that the notification of antenna lock
8535         failure changes has been enabled.
8536
8537         'false' Indicates that the notification of antenna lock
8538         failure changes has been disabled."
8539     ::= { epcgAntReadPointEntry 20 }
8540
8541 epcgAntRdPntLockFailuresNotifLevel OBJECT-TYPE
8542     SYNTAX      EpcgNotifLevel
8543     MAX-ACCESS  read-write
8544     STATUS      current
8545     DESCRIPTION
8546         "The severity level assigned to the
8547         epcgReaderAntennaLockFailure notification."
8548     DEFVAL { notice }
8549     ::= { epcgAntReadPointEntry 21 }
8550
8551 epcgAntRdPntLockFailureSuppressInterval OBJECT-TYPE
8552     SYNTAX      Unsigned32 ( 0 | 1..3600 )
8553     UNITS       "seconds"
8554     MAX-ACCESS  read-write
8555     STATUS      current
8556     DESCRIPTION
8557         "The length of the interval used to suppress generation
8558         of epcgReaderAntennaLockFailure notifications.
8559
8560         Notification will not be suppressed when zero value is
8561         specified."
8562     DEFVAL { 0 }
8563     ::= { epcgAntReadPointEntry 34 }
8564
8565 epcgAntRdPntLockFailureSuppressions OBJECT-TYPE
8566     SYNTAX      Counter32
8567     UNITS       "notifications"
8568     MAX-ACCESS  read-only
8569     STATUS      current
8570     DESCRIPTION
8571         "The number of epcgReaderAntennaLockFailure
8572         notifications that have been suppressed."

```

```

8573 ::= { epcgAntReadPointEntry 35 }
8574
8575 epcgAntRdPntPowerLevel OBJECT-TYPE
8576 SYNTAX Integer32(-2147483648..2147483647)
8577 MAX-ACCESS read-only
8578 STATUS current
8579 DESCRIPTION
8580 "The power level of this antenna read point.
8581 The information provided by this object will be in
8582 different formats depending on the value of the
8583 epcgRdrDevNormalizePowerLevel object. The different
8584 formats are provide in the describe for the
8585 epcgRdrDevNormalizePowerLevel object."
8586 ::= { epcgAntReadPointEntry 22 }
8587
8588 epcgAntRdPntNoiseLevel OBJECT-TYPE
8589 SYNTAX Integer32(-2147483648..2147483647)
8590 MAX-ACCESS read-only
8591 STATUS current
8592 DESCRIPTION
8593 "The noise level of this antenna read point.
8594 The information provided by this object will be in
8595 different formats depending on the value of the
8596 epcgRdrDevNormalizePowerLevel object. The different
8597 formats are provide in the describe for the
8598 epcgRdrDevNormalizePowerLevel object."
8599 ::= { epcgAntReadPointEntry 23 }
8600
8601 epcgAntRdPntTimeEnergized OBJECT-TYPE
8602 SYNTAX EpcgCurrentCountSinceReset
8603 UNITS "milli-seconds"
8604 MAX-ACCESS read-only
8605 STATUS current
8606 DESCRIPTION
8607 "The amount of time ,in milli-seconds, that this antenna
8608 read point since system started or value
8609 was cleared."
8610 ::= { epcgAntReadPointEntry 24 }
8611
8612 --*****
8613 --** IO Ports
8614 --*****
8615 epcgIoPortTable OBJECT-TYPE
8616 SYNTAX SEQUENCE OF EpcgIoPortEntry
8617 MAX-ACCESS not-accessible
8618 STATUS current
8619 DESCRIPTION
8620 "There is a row entry in this table for each data IO port
8621 that the reader supports. Typically these data IO ports
8622 send and/or receive event triggers."
8623 ::= { epcgIoPorts 1 }
8624
8625 epcgIoPortEntry OBJECT-TYPE
8626 SYNTAX EpcgIoPortEntry
8627 MAX-ACCESS not-accessible
8628 STATUS current
8629 DESCRIPTION
8630 "The properties of a single IO port. SNMP operations
8631 can neither create nor delete rows in the epcgIoPortTable"
8632 INDEX { epcgIoPortIndex }
8633 ::= { epcgIoPortTable 1 }
8634

```



```

8635 EpcgIoPortEntry ::= SEQUENCE {
8636     epcgIoPortIndex           Unsigned32,
8637     epcgIoPortName           SnmpAdminString,
8638     epcgIoPortAdminStatus    EpcgAdministrativeStatus,
8639     epcgIoPortOperStatus     EpcgOperationalStatus,
8640     epcgIoPortOperStatusNotifEnable TruthValue,
8641     epcgIoPortOperStatusNotifLevel EpcgNotifLevel,
8642     epcgIoPortOperStatusNotifFromState EpcgOperationalEnable,
8643     epcgIoPortOperStatusNotifToState EpcgOperationalEnable,
8644     epcgIoPortDescription    SnmpAdminString,
8645     epcgIoPortOperStatusPrior EpcgOperationalStatus,
8646     epcgIoPortOperStateSuppressInterval Unsigned32,
8647     epcgIoPortOperStateSuppressions Counter32
8648 }
8649
8650 epcgIoPortIndex OBJECT-TYPE
8651     SYNTAX      Unsigned32(1..2147483647)
8652     MAX-ACCESS  not-accessible
8653     STATUS      current
8654     DESCRIPTION
8655         "Index used to uniquely identifying an IO port within
8656         the scope of a given RFID Reader. This index is assigned
8657         arbitrarily by the SNMP engine and is not required to
8658         be saved over restarts."
8659     ::= { epcgIoPortEntry 1 }
8660
8661 epcgIoPortName OBJECT-TYPE
8662     SYNTAX      SnmpAdminString
8663     MAX-ACCESS  read-only
8664     STATUS      current
8665     DESCRIPTION
8666         "The name of this IO Port."
8667     ::= { epcgIoPortEntry 2 }
8668
8669 epcgIoPortAdminStatus OBJECT-TYPE
8670     SYNTAX      EpcgAdministrativeStatus
8671     MAX-ACCESS  read-write
8672     STATUS      current
8673     DESCRIPTION
8674         "The administratively specified status of the IO port."
8675     ::= { epcgIoPortEntry 3 }
8676
8677 epcgIoPortOperStatus OBJECT-TYPE
8678     SYNTAX      EpcgOperationalStatus
8679     MAX-ACCESS  read-only
8680     STATUS      current
8681     DESCRIPTION
8682         "The read point's current operational status."
8683     ::= { epcgIoPortEntry 4 }
8684
8685 epcgIoPortOperStatusNotifEnable OBJECT-TYPE
8686     SYNTAX      TruthValue
8687     MAX-ACCESS  read-write
8688     STATUS      current
8689     DESCRIPTION
8690         "This object controls the generation of the
8691         epcgReaderIoPortOperationState notification
8692         as follows.
8693
8694         'true' Indicates that the notification of IO port operation
8695         state changes has been enabled.
8696

```

```

8697         'false' Indicates that the notification of IO port operation
8698             state changes has been disabled."
8699         ::= { epcgIoPortEntry 5 }
8700
8701 epcgIoPortOperStatusNotifLevel OBJECT-TYPE
8702     SYNTAX      EpcgNotifLevel
8703     MAX-ACCESS  read-write
8704     STATUS      current
8705     DESCRIPTION
8706         "The severity level assigned to the
8707             epcgReaderDeviceOperationState notification"
8708     DEFVAL { error }
8709     ::= { epcgIoPortEntry 6 }
8710
8711 epcgIoPortOperStatusNotifFromState OBJECT-TYPE
8712     SYNTAX      EpcgOperationalEnable
8713     MAX-ACCESS  read-write
8714     STATUS      current
8715     DESCRIPTION
8716         "The source state for state transitions triggering
8717             epcgReaderIoPortOperationState notifications"
8718     DEFVAL { { unknown, other, up, down } }
8719     ::= { epcgIoPortEntry 7 }
8720
8721 epcgIoPortOperStatusNotifToState OBJECT-TYPE
8722     SYNTAX      EpcgOperationalEnable
8723     MAX-ACCESS  read-write
8724     STATUS      current
8725     DESCRIPTION
8726         "The destination state for state transitions triggering
8727             epcgReaderIoPortOperationState notifications"
8728     DEFVAL { { unknown, other, up, down } }
8729     ::= { epcgIoPortEntry 8 }
8730
8731 epcgIoPortDescription OBJECT-TYPE
8732     SYNTAX      SnmpAdminString
8733     MAX-ACCESS  read-only
8734     STATUS      current
8735     DESCRIPTION
8736         "The operator's textual description of the IO Port."
8737     ::= { epcgIoPortEntry 9 }
8738
8739 epcgIoPortOperStatusPrior OBJECT-TYPE
8740     SYNTAX      EpcgOperationalStatus
8741     MAX-ACCESS  accessible-for-notify
8742     STATUS      current
8743     DESCRIPTION
8744         "Status of IO port prior to status change that
8745             triggered THIS epcgReaderIoPortOperationState."
8746     ::= { epcgIoPortEntry 10 }
8747
8748 epcgIoPortOperStateSuppressInterval OBJECT-TYPE
8749     SYNTAX      Unsigned32 ( 0 | 1..3600 )
8750     UNITS       "seconds"
8751     MAX-ACCESS  read-write
8752     STATUS      current
8753     DESCRIPTION
8754         "The length of the interval used to suppress generation
8755             of epcgReaderIoPortOperationState change notifications.
8756
8757             Notification will not be suppressed when zero value is
8758             specified."

```

```

8759     DEFVAL { 0 }
8760     ::= { epcgIoPortEntry 11 }
8761
8762 epcgIoPortOperStateSuppressions OBJECT-TYPE
8763     SYNTAX          Counter32
8764     UNITS           "notifications"
8765     MAX-ACCESS     read-only
8766     STATUS         current
8767     DESCRIPTION
8768         "The number of IO port state change
8769         notifications that have been suppressed."
8770     ::= { epcgIoPortEntry 12 }
8771
8772 --*****
8773 --** (Logical) Sources (of tag data)
8774 --*****
8775 epcgSourceTable OBJECT-TYPE
8776     SYNTAX          SEQUENCE OF EpcgSourceEntry
8777     MAX-ACCESS     not-accessible
8778     STATUS         current
8779     DESCRIPTION
8780         "This table contains a row for each Source configured
8781         within a Reader.  A Source is a logical entity with
8782         zero, one, or more ReadPoints assigned to it.  A
8783         Source object may be associated with zero, one or
8784         more ReadTriggers and zero, one or more Notification
8785         Channels."
8786     ::= { epcgSources 1 }
8787
8788 epcgSourceEntry OBJECT-TYPE
8789     SYNTAX          EpcgSourceEntry
8790     MAX-ACCESS     not-accessible
8791     STATUS         current
8792     DESCRIPTION
8793         "The properties of a tag data source.  SNMP operations can
8794         neither create nor delete rows of this table."
8795
8796     INDEX { epcgSrcIndex }
8797     ::= { epcgSourceTable 1 }
8798
8799 EpcgSourceEntry ::= SEQUENCE {
8800     epcgSrcIndex          Unsigned32,
8801     epcgSrcName           SnmpAdminString,
8802     epcgSrcReadCyclesPerTrigger Unsigned32,
8803     epcgSrcReadDutyCycle Gauge32,
8804     epcgSrcReadTimeout   Unsigned32,
8805     epcgSrcGlimpsedTimeout Unsigned32,
8806     epcgSrcObservedThreshold Unsigned32,
8807     epcgSrcObservedTimeout Unsigned32,
8808     epcgSrcLostTimeout   Unsigned32,
8809     epcgSrcUnknownToGlimpsedTrans EpcgCurrentCountSinceReset,
8810     epcgSrcGlimpsedToUnknownTrans EpcgCurrentCountSinceReset,
8811     epcgSrcGlimpsedToObservedTrans EpcgCurrentCountSinceReset,
8812     epcgSrcObservedToLostTrans     EpcgCurrentCountSinceReset,
8813     epcgSrcLostToGlimpsedTrans     EpcgCurrentCountSinceReset,
8814     epcgSrcLostToUnknownTrans       EpcgCurrentCountSinceReset,
8815     epcgSrcAdminStatus              EpcgAdministrativeStatus,
8816     epcgSrcOperStatus               EpcgOperationalStatus,
8817     epcgSrcOperStatusNotifEnable    TruthValue,
8818     epcgSrcOperStatusNotifFromState EpcgOperationalEnable,
8819     epcgSrcOperStatusNotifToState   EpcgOperationalEnable,
8820     epcgSrcOperStatusNotifyLevel    EpcgNotifLevel,

```

```

8821         epcgSrcSupportsWriteOperations TruthValue,
8822         epcgSrcOperStatusPrior         EpcgOperationalStatus,
8823         epcgSrcOperStateSupprssInterval Unsigned32,
8824         epcgSrcOperStateSuppressions   Counter32
8825     }
8826
8827 epcgSrcIndex OBJECT-TYPE
8828     SYNTAX      Unsigned32(1..2147483647)
8829     MAX-ACCESS  not-accessible
8830     STATUS      current
8831     DESCRIPTION
8832         "Index used for uniquely identifying a source within the
8833         scope of a given RFID reader. This index is assigned
8834         arbitrarily by the SNMP engine and is not required to
8835         be saved over restarts."
8836     ::= { epcgSourceEntry 1 }
8837
8838 epcgSrcName OBJECT-TYPE
8839     SYNTAX      SnmpAdminString
8840     MAX-ACCESS  read-only
8841     STATUS      current
8842     DESCRIPTION
8843         "Unique name assigned to a source."
8844     ::= { epcgSourceEntry 2 }
8845
8846 epcgSrcReadCyclesPerTrigger OBJECT-TYPE
8847     SYNTAX      Unsigned32
8848     UNITS       "cycles"
8849     MAX-ACCESS  read-write
8850     STATUS      current
8851     DESCRIPTION
8852         "Read cycles attempted per trigger."
8853     ::= { epcgSourceEntry 3 }
8854
8855 epcgSrcReadDutyCycle OBJECT-TYPE
8856     SYNTAX      Gauge32(0..100)
8857     UNITS       "percentage"
8858     MAX-ACCESS  read-write
8859     STATUS      current
8860     DESCRIPTION
8861         "Duty cycle for this read source."
8862     ::= { epcgSourceEntry 4 }
8863
8864 epcgSrcReadTimeout OBJECT-TYPE
8865     SYNTAX      Unsigned32
8866     UNITS       "milli-seconds"
8867     MAX-ACCESS  read-write
8868     STATUS      current
8869     DESCRIPTION
8870         "Read timeout in milli-seconds for this source."
8871     ::= { epcgSourceEntry 5 }
8872
8873 epcgSrcGlimpsedTimeout OBJECT-TYPE
8874     SYNTAX      Unsigned32
8875     UNITS       "milli-seconds"
8876     MAX-ACCESS  read-write
8877     STATUS      current
8878     DESCRIPTION
8879         "Glimpsed timeout in milli-seconds for this source."
8880     ::= { epcgSourceEntry 6 }
8881
8882 epcgSrcObservedThreshold OBJECT-TYPE

```

```

8883 SYNTAX          Unsigned32
8884 UNITS          "milli-seconds"
8885 MAX-ACCESS      read-write
8886 STATUS          current
8887 DESCRIPTION
8888     "Number of milli-seconds required for tag to be considered
8889     observed for this source"
8890 ::= { epcgSourceEntry 7 }
8891
8892 epcgSrcObservedTimeout OBJECT-TYPE
8893 SYNTAX          Unsigned32
8894 UNITS          "milli-seconds"
8895 MAX-ACCESS      read-write
8896 STATUS          current
8897 DESCRIPTION
8898     "Number of milli-seconds required for tag to be considered
8899     observed for this source"
8900 ::= { epcgSourceEntry 8 }
8901
8902 epcgSrcLostTimeout OBJECT-TYPE
8903 SYNTAX          Unsigned32
8904 UNITS          "milli-seconds"
8905 MAX-ACCESS      read-write
8906 STATUS          current
8907 DESCRIPTION
8908     "Number of milli-seconds required for tag to be considered lost
8909     for this source"
8910 ::= { epcgSourceEntry 9 }
8911
8912 epcgSrcUnknownToGlimpsedTrans OBJECT-TYPE
8913 SYNTAX          EpcgCurrentCountSinceReset
8914 UNITS          "transitions"
8915 MAX-ACCESS      read-only
8916 STATUS          current
8917 DESCRIPTION
8918     "Number of tags transitioning from 'unknown' status to
8919     'glimpsed' status for this source."
8920 ::= { epcgSourceEntry 10 }
8921
8922 epcgSrcGlimpsedToUnknownTrans OBJECT-TYPE
8923 SYNTAX          EpcgCurrentCountSinceReset
8924 UNITS          "transitions"
8925 MAX-ACCESS      read-only
8926 STATUS          current
8927 DESCRIPTION
8928     "Number of tags transitioning from 'glimpsed' status to
8929     'unknown' status for this source."
8930 ::= { epcgSourceEntry 11 }
8931
8932 epcgSrcGlimpsedToObservedTrans OBJECT-TYPE
8933 SYNTAX          EpcgCurrentCountSinceReset
8934 UNITS          "transitions"
8935 MAX-ACCESS      read-only
8936 STATUS          current
8937 DESCRIPTION
8938     "Number of tags transitioning from 'glimpsed' status
8939     to 'observed' status for this source."
8940 ::= { epcgSourceEntry 12 }
8941
8942 epcgSrcObservedToLostTrans OBJECT-TYPE
8943 SYNTAX          EpcgCurrentCountSinceReset
8944 UNITS          "transitions"

```

```

8945 MAX-ACCESS      read-only
8946 STATUS        current
8947 DESCRIPTION
8948     "Number of tags transitioning from 'observed' status
8949     to 'lost' status for this source."
8950 ::= { epcgSourceEntry 13 }
8951
8952 epcgSrcLostToGlimpsedTrans OBJECT-TYPE
8953 SYNTAX        EpcgCurrentCountSinceReset
8954 UNITS         "transitions"
8955 MAX-ACCESS    read-only
8956 STATUS        current
8957 DESCRIPTION
8958     "Number of tags transitioning from 'lost' status to
8959     'glimpsed' status for this source"
8960 ::= { epcgSourceEntry 14 }
8961
8962 epcgSrcLostToUnknownTrans OBJECT-TYPE
8963 SYNTAX        EpcgCurrentCountSinceReset
8964 MAX-ACCESS    read-only
8965 STATUS        current
8966 DESCRIPTION
8967     "Number of tags transitioning from 'lost' status to
8968     'unknown' status for this source"
8969 ::= { epcgSourceEntry 15 }
8970
8971 epcgSrcAdminStatus OBJECT-TYPE
8972 SYNTAX        EpcgAdministrativeStatus
8973 MAX-ACCESS    read-write
8974 STATUS        current
8975 DESCRIPTION
8976     "The administratively specified status of the source."
8977 ::= { epcgSourceEntry 16 }
8978
8979 epcgSrcOperStatus OBJECT-TYPE
8980 SYNTAX        EpcgOperationalStatus
8981 MAX-ACCESS    read-only
8982 STATUS        current
8983 DESCRIPTION
8984     "The source's current operational status."
8985 ::= { epcgSourceEntry 17 }
8986
8987 epcgSrcOperStatusNotifEnable OBJECT-TYPE
8988 SYNTAX        TruthValue
8989 MAX-ACCESS    read-write
8990 STATUS        current
8991 DESCRIPTION
8992     "This object controls the generation of the
8993     epcgReaderSourceOperationState notification
8994     as follows.
8995
8996     'true' Indicates that the notification of Source state
8997     changes has been enabled.
8998
8999     'false' Indicates that the notification of Source state
9000     changes has been disabled."
9001 ::= { epcgSourceEntry 18 }
9002
9003 epcgSrcOperStatusNotifFromState OBJECT-TYPE
9004 SYNTAX        EpcgOperationalEnable
9005 MAX-ACCESS    read-write
9006 STATUS        current

```

```

9007 DESCRIPTION
9008     "The source state for state transitions triggering
9009     epcgReaderSourceOperationState notifications"
9010 DEFVAL { { unknown, other, up, down } }
9011 ::= { epcgSourceEntry 19 }
9012
9013 epcgSrcOperStatusNotifToState OBJECT-TYPE
9014 SYNTAX          EpcgOperationalEnable
9015 MAX-ACCESS      read-write
9016 STATUS          current
9017 DESCRIPTION
9018     "The destination state for state transitions triggering
9019     epcgReaderSourceOperationState notifications"
9020 DEFVAL { { unknown, other, up, down } }
9021 ::= { epcgSourceEntry 20 }
9022
9023 epcgSrcOperStatusNotifyLevel OBJECT-TYPE
9024 SYNTAX          EpcgNotifLevel
9025 MAX-ACCESS      read-write
9026 STATUS          current
9027 DESCRIPTION
9028     "The severity level assigned to the
9029     epcgReaderSourceOperationState notification."
9030 DEFVAL { error }
9031 ::= { epcgSourceEntry 21 }
9032
9033 epcgSrcSupportsWriteOperations OBJECT-TYPE
9034 SYNTAX          TruthValue
9035 MAX-ACCESS      read-only
9036 STATUS          current
9037 DESCRIPTION
9038     "This object provides an indication of the capability of the
9039     source as follows.
9040     'true' - source supports read and write operations.
9041     'false' - source supports only read operations."
9042 ::= { epcgSourceEntry 22 }
9043
9044 epcgSrcOperStatusPrior OBJECT-TYPE
9045 SYNTAX          EpcgOperationalStatus
9046 MAX-ACCESS      accessible-for-notify
9047 STATUS          current
9048 DESCRIPTION
9049     "Status of EPC data source prior to status change that
9050     triggered the epcgReaderSourceOperationState
9051     notification."
9052 ::= { epcgSourceEntry 23 }
9053
9054 epcgSrcOperStateSuppressInterval OBJECT-TYPE
9055 SYNTAX          Unsigned32 ( 0 | 1..3600 )
9056 UNITS           "seconds"
9057 MAX-ACCESS      read-write
9058 STATUS          current
9059 DESCRIPTION
9060     "The length of the interval used to suppress generation
9061     of epcgReaderSourceOperationState change notifications.
9062
9063     Notification will not be suppressed when zero value is
9064     specified."
9065 DEFVAL { 0 }
9066 ::= { epcgSourceEntry 24 }
9067
9068 epcgSrcOperStateSuppressions OBJECT-TYPE

```

```

9069     SYNTAX          Counter32
9070     UNITS            "notifications"
9071     MAX-ACCESS       read-only
9072     STATUS           current
9073     DESCRIPTION
9074         "The number of source operational state change
9075         notifications that have been suppressed."
9076     ::= { epcgSourceEntry 25 }
9077
9078     --*****
9079     --** Notification Channels
9080     --*****
9081     epcgNotificationChannelTable OBJECT-TYPE
9082         SYNTAX          SEQUENCE OF EpcgNotificationChannelEntry
9083         MAX-ACCESS       not-accessible
9084         STATUS           current
9085         DESCRIPTION
9086             "This table contains one row per notification channel.
9087             A notification channel is associated with zero, one
9088             or more Sources, and zero, one Notification Triggers"
9089     ::= { epcgNotificationChannels 1 }
9090
9091     epcgNotificationChannelEntry OBJECT-TYPE
9092         SYNTAX          EpcgNotificationChannelEntry
9093         MAX-ACCESS       not-accessible
9094         STATUS           current
9095         DESCRIPTION
9096             "The properties of a notification channel.  SNMP operations
9097             can neither create nor delete rows of this table"
9098         INDEX { epcgNotifChanIndex }
9099     ::= { epcgNotificationChannelTable 1 }
9100
9101     EpcgNotificationChannelEntry ::= SEQUENCE {
9102         epcgNotifChanIndex      Unsigned32,
9103         epcgNotifChanName       SnmpAdminString,
9104         epcgNotifChanAddressType InetAddressType,
9105         epcgNotifChanAddress    InetAddress,
9106         epcgNotifChanLastAttempt DateAndTime,
9107         epcgNotifChanLastSuccess DateAndTime,
9108         epcgNotifChanAdminStatus EpcgAdministrativeStatus,
9109         epcgNotifChanOperStatus EpcgOperationalStatus,
9110         epcgNotifChanOperNotifEnable TruthValue,
9111         epcgNotifChanOperNotifLevel EpcgNotifLevel,
9112         epcgNotifChanOperNotifFromState EpcgOperationalEnable,
9113         epcgNotifChanOperNotifToState EpcgOperationalEnable,
9114         epcgNotifChanOperStatusPrior EpcgOperationalStatus,
9115         epcgNotifChanOperStateSuppressInterval Unsigned32,
9116         epcgNotifChanOperStateSuppressions Counter32
9117     }
9118
9119     epcgNotifChanIndex OBJECT-TYPE
9120         SYNTAX          Unsigned32(1..2147483647)
9121         MAX-ACCESS       not-accessible
9122         STATUS           current
9123         DESCRIPTION
9124             "Index used for uniquely identifying a notification channel
9125             within the scope of a given RFID Reader.  This index is
9126             assigned arbitrarily by the SNMP engine and is not required
9127             to be saved over restarts."
9128     ::= { epcgNotificationChannelEntry 1 }
9129
9130     epcgNotifChanName OBJECT-TYPE

```



```

9131     SYNTAX          SnmpAdminString
9132     MAX-ACCESS      read-only
9133     STATUS          current
9134     DESCRIPTION
9135         "Unique name assigned to Notification Channel."
9136     ::= { epcgNotificationChannelEntry 2 }
9137
9138     epcgNotifChanAddressType OBJECT-TYPE
9139     SYNTAX          InetAddressType
9140     MAX-ACCESS      read-only
9141     STATUS          current
9142     DESCRIPTION
9143         "The type of Internet address used to identify the
9144         notification channel's destination"
9145     ::= { epcgNotificationChannelEntry 3 }
9146
9147     epcgNotifChanAddress OBJECT-TYPE
9148     SYNTAX          InetAddress
9149     MAX-ACCESS      read-only
9150     STATUS          current
9151     DESCRIPTION
9152         "The Internet address identifying the notification
9153         channel's destination."
9154     ::= { epcgNotificationChannelEntry 4 }
9155
9156     epcgNotifChanLastAttempt OBJECT-TYPE
9157     SYNTAX          DateAndTime
9158     MAX-ACCESS      read-only
9159     STATUS          current
9160     DESCRIPTION
9161         "Wall-clock time of last attempted notification by this
9162         notification channel."
9163     ::= { epcgNotificationChannelEntry 5 }
9164
9165     epcgNotifChanLastSuccess OBJECT-TYPE
9166     SYNTAX          DateAndTime
9167     MAX-ACCESS      read-only
9168     STATUS          current
9169     DESCRIPTION
9170         "Wall-clock time of last successful notification sent by
9171         this notification channel."
9172     ::= { epcgNotificationChannelEntry 6 }
9173
9174     epcgNotifChanAdminStatus OBJECT-TYPE
9175     SYNTAX          EpcgAdministrativeStatus
9176     MAX-ACCESS      read-write
9177     STATUS          current
9178     DESCRIPTION
9179         "The administratively specified status of the
9180         notification channel."
9181     ::= { epcgNotificationChannelEntry 7 }
9182
9183     epcgNotifChanOperStatus OBJECT-TYPE
9184     SYNTAX          EpcgOperationalStatus
9185     MAX-ACCESS      read-only
9186     STATUS          current
9187     DESCRIPTION
9188         "The notification channel's current operational status."
9189     ::= { epcgNotificationChannelEntry 8 }
9190
9191     epcgNotifChanOperNotifEnable OBJECT-TYPE
9192     SYNTAX          TruthValue

```

```

9193 MAX-ACCESS      read-write
9194 STATUS         current
9195 DESCRIPTION
9196     "This object controls the generation of the
9197     epcgReaderNotificationChanOperState notification
9198     as follows.
9199
9200     'true' - Indicates that the sending of notification channel
9201             operation state changes has been enabled.
9202
9203     'false' - Indicates that the sending of notification channel
9204              operation state changes has been disabled."
9205 ::= { epcgNotificationChannelEntry 9 }
9206
9207 epcgNotifChanOperNotifLevel OBJECT-TYPE
9208 SYNTAX          EpcgNotifLevel
9209 MAX-ACCESS      read-write
9210 STATUS         current
9211 DESCRIPTION
9212     "The severity level assigned to the
9213     epcgReaderNotificationChanOperState notification."
9214 DEFVAL { error }
9215 ::= { epcgNotificationChannelEntry 10 }
9216
9217 epcgNotifChanOperNotifFromState OBJECT-TYPE
9218 SYNTAX          EpcgOperationalEnable
9219 MAX-ACCESS      read-write
9220 STATUS         current
9221 DESCRIPTION
9222     "The source state for the state transitions triggering
9223     epcgReaderNotificationChanOperState notificatons"
9224 DEFVAL { { unknown, other, up, down} }
9225 ::= { epcgNotificationChannelEntry 11 }
9226
9227 epcgNotifChanOperNotifToState OBJECT-TYPE
9228 SYNTAX          EpcgOperationalEnable
9229 MAX-ACCESS      read-write
9230 STATUS         current
9231 DESCRIPTION
9232     "The destination state for the state transitions triggering
9233     epcgReaderNotificationChanOperState notifications"
9234 DEFVAL { { unknown, other, up, down} }
9235 ::= { epcgNotificationChannelEntry 12 }
9236
9237 epcgNotifChanOperStatusPrior OBJECT-TYPE
9238 SYNTAX          EpcgOperationalStatus
9239 MAX-ACCESS      accessible-for-notify
9240 STATUS         current
9241 DESCRIPTION
9242     "The status of notification channel prior to the current
9243     status change that triggered the
9244     epcgReaderNotificationChanOperState notification"
9245 ::= { epcgNotificationChannelEntry 13 }
9246
9247 epcgNotifChanOperStateSuppressInterval OBJECT-TYPE
9248 SYNTAX          Unsigned32 ( 0 | 1..3600 )
9249 UNITS           "seconds"
9250 MAX-ACCESS      read-write
9251 STATUS         current
9252 DESCRIPTION
9253     "The length of the interval used to suppress generation
9254     of epcgReaderNotificationChanOperState change notifications.

```

```

9255
9256         Notification will not be suppressed when zero value is
9257         specified."
9258     DEFVAL { 0 }
9259     ::= { epcgNotificationChannelEntry 14 }
9260
9261 epcgNotifChanOperStateSuppressions OBJECT-TYPE
9262     SYNTAX          Counter32
9263     UNITS           "notifications"
9264     MAX-ACCESS      read-only
9265     STATUS          current
9266     DESCRIPTION
9267         "The number of channel operational state change
9268         notifications that have been suppressed."
9269     ::= { epcgNotificationChannelEntry 15 }
9270
9271 --*****
9272 --** Triggers
9273 --*****
9274 epcgTriggerTable OBJECT-TYPE
9275     SYNTAX          SEQUENCE OF EpcgTriggerEntry
9276     MAX-ACCESS      not-accessible
9277     STATUS          current
9278     DESCRIPTION
9279         "This table contains one row per trigger. A
9280         Trigger is associated with zero or one IO Port.
9281         A Trigger may be associated with zero or more Sources
9282         and zero or more Notification Channels"
9283     ::= { epcgTriggers 1 }
9284
9285 epcgTriggerEntry OBJECT-TYPE
9286     SYNTAX          EpcgTriggerEntry
9287     MAX-ACCESS      not-accessible
9288     STATUS          current
9289     DESCRIPTION
9290         "The properties of a trigger. SNMP operations can neither
9291         create nor delete rows in the epcgTriggerTable."
9292     INDEX { epcgTrigIndex }
9293     ::= { epcgTriggerTable 1 }
9294
9295 EpcgTriggerEntry ::= SEQUENCE {
9296     epcgTrigIndex      Unsigned32,
9297     epcgTrigName       SnmpAdminString,
9298     epcgTrigType       EpcgTriggerType,
9299     epcgTrigParameters SnmpAdminString,
9300     epcgTriggerMatches EpcgCurrentCountSinceReset,
9301     epcgTrigIoPort     RowPointer
9302 }
9303
9304 epcgTrigIndex OBJECT-TYPE
9305     SYNTAX          Unsigned32(1..2147483647)
9306     MAX-ACCESS      not-accessible
9307     STATUS          current
9308     DESCRIPTION
9309         "Index uniquely identifying this trigger. This index is
9310         assigned arbitrarily by the SNMP engine and is not
9311         required to be saved over restarts."
9312     ::= { epcgTriggerEntry 1 }
9313
9314 epcgTrigName OBJECT-TYPE
9315     SYNTAX          SnmpAdminString
9316     MAX-ACCESS      read-only

```

```

9317 STATUS current
9318 DESCRIPTION
9319 "Name of this trigger."
9320 ::= { epcgTriggerEntry 2 }
9321
9322 epcgTrigType OBJECT-TYPE
9323 SYNTAX EpcgTriggerType
9324 MAX-ACCESS read-only
9325 STATUS current
9326 DESCRIPTION
9327 "The trigger type."
9328 ::= { epcgTriggerEntry 3 }
9329
9330 epcgTrigParameters OBJECT-TYPE
9331 SYNTAX SnmpAdminString
9332 MAX-ACCESS read-only
9333 STATUS current
9334 DESCRIPTION
9335 "The epcgTrigParameters value must comply with the constraints
9336 defined for the 'triggervalue' datatype in the EPCglobal
9337 Reader Protocol Specification, 1.1 or greater."
9338 REFERENCE
9339 "EPCglobal Reader Protocol Specification section 6.4"
9340 ::= { epcgTriggerEntry 4 }
9341
9342 epcgTriggerMatches OBJECT-TYPE
9343 SYNTAX EpcgCurrentCountSinceReset
9344 UNITS "occurrences"
9345 MAX-ACCESS read-only
9346 STATUS current
9347 DESCRIPTION
9348 "Number of times this trigger has matched."
9349 ::= { epcgTriggerEntry 5 }
9350
9351 epcgTrigIoPort OBJECT-TYPE
9352 SYNTAX RowPointer
9353 MAX-ACCESS read-only
9354 STATUS current
9355 DESCRIPTION
9356 "A pointer to an entry in the IoPort table. A
9357 trigger need not correspond to an IO port
9358 (e.g., a trigger of type timer has not IO port
9359 association), in which case this
9360 RowPointer object has the value zeroDotzero"
9361 ::= { epcgTriggerEntry 6 }
9362
9363 --*****
9364 --** Notification Trigger Table
9365 --*****
9366 epcgNotifTrigTable OBJECT-TYPE
9367 SYNTAX SEQUENCE OF EpcgNotifTrigEntry
9368 MAX-ACCESS not-accessible
9369 STATUS current
9370 DESCRIPTION
9371 "This table associates triggers with notification channels.
9372 Multiple triggers may be associated with the same
9373 notification channel and a single trigger may be associated
9374 with multiple notification channels. There will be one
9375 row entry for each unique pairing of trigger and notification
9376 channel. "
9377 ::= { epcgTriggers 2 }
9378

```

```

9379 epcgNotifTrigEntry OBJECT-TYPE
9380     SYNTAX          EpcgNotifTrigEntry
9381     MAX-ACCESS      not-accessible
9382     STATUS          current
9383     DESCRIPTION
9384         "Identifies a single pairing of a trigger and notification
9385         channel.  Entries can be added and removed from this
9386         table via epcgNotifTrigRowStatus in accordance with the
9387         RowStatus convention."
9388         INDEX { epcgNotifChanIndex,
9389                epcgTrigIndex }
9390     ::= { epcgNotifTrigTable 1 }
9391
9392 EpcgNotifTrigEntry ::= SEQUENCE {
9393     epcgNotifTrigRowStatus      RowStatus
9394 }
9395
9396 epcgNotifTrigRowStatus OBJECT-TYPE
9397     SYNTAX          RowStatus
9398     MAX-ACCESS      read-create
9399     STATUS          current
9400     DESCRIPTION
9401         "The status of this row."
9402     ::= { epcgNotifTrigEntry 1 }
9403
9404 --*****
9405 --** Read Trigger Table
9406 --*****
9407 epcgReadTrigTable OBJECT-TYPE
9408     SYNTAX          SEQUENCE OF EpcgReadTrigEntry
9409     MAX-ACCESS      not-accessible
9410     STATUS          current
9411     DESCRIPTION
9412         "This table associates Sources with Triggers.  Multiple
9413         triggers may be associated with the same Source and a
9414         single trigger may be associated with multiple Sources.
9415         There will be one row entry for each unique pairing of
9416         Trigger and Source.  Entries can be added and removed from
9417         this table via epcgReadTrigRowStatus in accordance with the
9418         RowStatus convention."
9419     ::= { epcgTriggers 3 }
9420
9421 epcgReadTrigEntry OBJECT-TYPE
9422     SYNTAX          EpcgReadTrigEntry
9423     MAX-ACCESS      not-accessible
9424     STATUS          current
9425     DESCRIPTION
9426         "Identifies a single pairing of a Source and Trigger."
9427         INDEX { epcgSrcIndex,
9428                epcgTrigIndex }
9429     ::= { epcgReadTrigTable 1 }
9430
9431 EpcgReadTrigEntry ::= SEQUENCE {
9432     epcgReadTrigRowStatus      RowStatus
9433 }
9434
9435 epcgReadTrigRowStatus OBJECT-TYPE
9436     SYNTAX          RowStatus
9437     MAX-ACCESS      read-create
9438     STATUS          current
9439     DESCRIPTION
9440         "The status of this row."

```

```

9441 ::= { epcgReadTrigEntry 1 }
9442
9443 --*****
9444 --** ReadPoint-Source Table
9445 --*****
9446 epcgRdPntSrcTable OBJECT-TYPE
9447     SYNTAX          SEQUENCE OF EpcgRdPntSrcEntry
9448     MAX-ACCESS      not-accessible
9449     STATUS          current
9450     DESCRIPTION
9451         "This table associates ReadPoints with (tag) Sources.
9452         Multiple ReadPoints may be associated with the same
9453         Source and a single ReadPoint may be associated with
9454         multiple Sources. There will be one row entry for
9455         each unique pairing of ReadPoint and Source"
9456     ::= { epcgSources 3 }
9457
9458 epcgRdPntSrcEntry OBJECT-TYPE
9459     SYNTAX          EpcgRdPntSrcEntry
9460     MAX-ACCESS      not-accessible
9461     STATUS          current
9462     DESCRIPTION
9463         "Identifies a single pairing of a Read Point and Source."
9464     INDEX { epcgReadPointIndex,
9465             epcgSrcIndex }
9466     ::= { epcgRdPntSrcTable 1 }
9467
9468 EpcgRdPntSrcEntry ::= SEQUENCE {
9469     epcgRdPntSrcRowStatus          RowStatus
9470 }
9471
9472 epcgRdPntSrcRowStatus OBJECT-TYPE
9473     SYNTAX          RowStatus
9474     MAX-ACCESS      read-create
9475     STATUS          current
9476     DESCRIPTION
9477         "The status of this row."
9478     ::= { epcgRdPntSrcEntry 1 }
9479
9480
9481 --*****
9482 --** Notification Channel - Source Table
9483 --*****
9484 epcgNotifChanSrcTable OBJECT-TYPE
9485     SYNTAX          SEQUENCE OF EpcgNotifChanSrcEntry
9486     MAX-ACCESS      not-accessible
9487     STATUS          current
9488     DESCRIPTION
9489         "This table associates Notification Channels with
9490         (tag) Sources. Multiple Notification Channels may be
9491         associated with the same Source and a single Notification
9492         Channel may be associated with multiple Sources. There
9493         will be one row entry for each unique pairing of
9494         Notification Channel and Source."
9495     ::= { epcgSources 4 }
9496
9497 epcgNotifChanSrcEntry OBJECT-TYPE
9498     SYNTAX          EpcgNotifChanSrcEntry
9499     MAX-ACCESS      not-accessible
9500     STATUS          current
9501     DESCRIPTION
9502         "Identifies a single pairing of a Notificaiton

```

```

9503         Channel and Source.
9504
9505         Entries can be added and removed from
9506         this table via epcgNotifChanSrcRowStatus in accordance with the
9507         RowStatus convention."
9508         INDEX { epcgNotifChanIndex,
9509                 epcgSrcIndex }
9510         ::= { epcgNotifChanSrcTable 1 }
9511
9512 EpcgNotifChanSrcEntry ::= SEQUENCE {
9513     epcgNotifChanSrcRowStatus      RowStatus
9514 }
9515
9516 epcgNotifChanSrcRowStatus OBJECT-TYPE
9517     SYNTAX                      RowStatus
9518     MAX-ACCESS                  read-create
9519     STATUS                      current
9520     DESCRIPTION
9521         "The status of this row."
9522     ::= { epcgNotifChanSrcEntry 1 }
9523
9524 --*****
9525 --* Conformance definitions
9526 --*****
9527 epcgReaderGroups      OBJECT IDENTIFIER ::= { epcgReaderConformance 1 }
9528 epcgReaderCompliances OBJECT IDENTIFIER ::= { epcgReaderConformance 2 }
9529
9530
9531 --*****
9532 --* Compliances statements
9533 --*****
9534 epcgReaderStandardCompliance MODULE-COMPLIANCE
9535     STATUS          current
9536     DESCRIPTION
9537         "This define the initial standard implementation requirements
9538         for an EPCglobal-compliant Reader.
9539
9540         The following groups are optional.
9541         epcgReaderGlobalMeasurementsGroup
9542         epcgReaderAntennaSuppressNotifyGroup
9543         epcgReaderNotifGroup
9544         epcgReaderMemoryGroup
9545         epcgReaderServerInfoGroup
9546         epcgReaderIOPortGroup
9547         epcgReaderNotifChannelGroup
9548         epcgReaderTriggerGroup
9549         epcgReaderTableAssocGroup
9550     "
9551     MODULE          -- This module
9552     MANDATORY-GROUPS { epcgReaderDeviceInformationGroup,
9553                       epcgReaderOperationGroup,
9554                       epcgReadPointGroup,
9555                       epcgReaderAntennaReadPointGroup,
9556                       epcgReaderSourceGroup,
9557                       epcgReaderMandatoryTableAssocGroup
9558                     }
9559
9560 -- *** epcgSourceTable *****
9561 OBJECT          epcgSrcReadCyclesPerTrigger
9562 MIN-ACCESS      read-only
9563 DESCRIPTION     "Write access is not required."
9564

```

```

9565 OBJECT          epcgSrcReadDutyCycle
9566 MIN-ACCESS      read-only
9567 DESCRIPTION     "Write access is not required."
9568
9569 OBJECT          epcgSrcReadTimeout
9570 MIN-ACCESS      read-only
9571 DESCRIPTION     "Write access is not required."
9572
9573 OBJECT          epcgSrcGlimpsedTimeout
9574 MIN-ACCESS      read-only
9575 DESCRIPTION     "Write access is not required."
9576
9577 OBJECT          epcgSrcObservedThreshold
9578 MIN-ACCESS      read-only
9579 DESCRIPTION     "Write access is not required."
9580
9581 OBJECT          epcgSrcObservedTimeout
9582 MIN-ACCESS      read-only
9583 DESCRIPTION     "Write access is not required."
9584
9585 OBJECT          epcgSrcLostTimeout
9586 MIN-ACCESS      read-only
9587 DESCRIPTION     "Write access is not required."
9588
9589 ::= { epcgReaderCompliances 1 }
9590
9591 --*****
9592 --** Units of conformance
9593 --*****
9594 epcgReaderNotifGroup NOTIFICATION-GROUP
9595     NOTIFICATIONS { epcgReaderDeviceOperationState,
9596                   epcgRdrDevMemoryState,
9597                   epcgReadPointOperationState,
9598                   epcgReaderAntennaReadFailure,
9599                   epcgReaderAntennaWriteFailure,
9600                   epcgReaderAntennaKillFailure,
9601                   epcgReaderAntennaEraseFailure,
9602                   epcgReaderAntennaLockFailure,
9603                   epcgReaderIoPortOperationState,
9604                   epcgReaderSourceOperationState,
9605                   epcgReaderNotificationChanOperState
9606                   }
9607     STATUS          current
9608     DESCRIPTION     "Reader device notifications."
9609     ::= { epcgReaderGroups 1 }
9610
9611
9612
9613 epcgReaderGlobalMeasurementsGroup OBJECT-GROUP
9614     OBJECTS        { epcgGlobalCountersData }
9615     STATUS          current
9616     DESCRIPTION     "Reader device global measurements objects."
9617     ::= { epcgReaderGroups 2 }
9618
9619
9620
9621 epcgReaderDeviceInformationGroup OBJECT-GROUP
9622     OBJECTS        { epcgRdrDevDescription,
9623                   epcgRdrDevRole,
9624                   epcgRdrDevEpc,
9625                   epcgRdrDevSerialNumber,
9626                   epcgRdrDevTimeUtc,

```



```

9627         epcgRdrDevCurrentSource,
9628         epcgRdrDevReboot,
9629         epcgRdrDevResetStatistics,
9630         epcgRdrDevResetTimestamp,
9631         epcgRdrDevNormalizePowerLevel,
9632         epcgRdrDevNormalizeNoiseLevel
9633     }
9634     STATUS      current
9635     DESCRIPTION
9636         "Reader device information objects"
9637     ::= { epcgReaderGroups 3 }
9638
9639
9640     epcgReaderOperationGroup OBJECT-GROUP
9641     OBJECTS      { epcgRdrDevOperStatus,
9642                   epcgRdrDevOperStatusPrior,
9643                   epcgRdrDevOperStateEnable,
9644                   epcgRdrDevOperNotifFromState,
9645                   epcgRdrDevOperNotifToState,
9646                   epcgRdrDevOperNotifStateLevel,
9647                   epcgRdrDevOperStateSuppressInterval,
9648                   epcgRdrDevOperStateSuppressions
9649     }
9650     STATUS      current
9651     DESCRIPTION
9652         "Reader device operation objects."
9653     ::= { epcgReaderGroups 4 }
9654
9655
9656     epcgReaderMemoryGroup OBJECT-GROUP
9657     OBJECTS      { epcgRdrDevFreeMemory,
9658                   epcgRdrDevFreeMemoryNotifEnable,
9659                   epcgRdrDevFreeMemoryNotifLevel,
9660                   epcgRdrDevFreeMemoryOnsetThreshold,
9661                   epcgRdrDevFreeMemoryAbateThreshold,
9662                   epcgRdrDevFreeMemoryStatus,
9663                   epcgRdrDevMemStateSuppressInterval,
9664                   epcgRdrDevMemStateSuppressions
9665     }
9666     STATUS      current
9667     DESCRIPTION
9668         "This mandatory group defines Device-level objects"
9669     ::= { epcgReaderGroups 5 }
9670
9671
9672     epcgReaderServerInfoGroup OBJECT-GROUP
9673     OBJECTS      { epcgReaderServerAddressType,
9674                   epcgReaderServerAddress,
9675                   epcgReaderServerRowStatus
9676     }
9677     STATUS      current
9678     DESCRIPTION
9679         "Reader device Server objects."
9680     ::= { epcgReaderGroups 6 }
9681
9682
9683     epcgReadPointGroup OBJECT-GROUP
9684     OBJECTS      { epcgReadPointName,
9685                   epcgReadPointDescription,
9686                   epcgReadPointAdminStatus,
9687                   epcgReadPointOperStatus,
9688                   epcgReadPointOperStateNotifyEnable,

```

```

9689         epcgReadPointOperNotifyFromState,
9690         epcgReadPointOperNotifyToState,
9691         epcgReadPointOperNotifyStateLevel,
9692         epcgReadPointOperStatusPrior,
9693         epcgReadPointOperStateSuppressInterval,
9694         epcgReadPointOperStateSuppressions
9695     }
9696     STATUS      current
9697     DESCRIPTION
9698         "Reader - Read Point Objects."
9699     ::= { epcgReaderGroups 7 }
9700
9701     epcgReaderAntennaReadPointGroup OBJECT-GROUP
9702     OBJECTS      { epcgAntRdPntTagsIdentified,
9703                   epcgAntRdPntTagsNotIdentified,
9704                   epcgAntRdPntMemoryReadFailures,
9705                   epcgAntRdPntReadFailureNotifEnable,
9706                   epcgAntRdPntReadFailureNotifLevel,
9707                   epcgAntRdPntWriteOperations,
9708                   epcgAntRdPntWriteFailures,
9709                   epcgAntRdPntWriteFailuresNotifEnable,
9710                   epcgAntRdPntWriteFailuresNotifLevel,
9711                   epcgAntRdPntKillOperations,
9712                   epcgAntRdPntKillFailures,
9713                   epcgAntRdPntKillFailuresNotifEnable,
9714                   epcgAntRdPntKillFailuresNotifLevel,
9715                   epcgAntRdPntEraseOperations,
9716                   epcgAntRdPntEraseFailures,
9717                   epcgAntRdPntEraseFailuresNotifEnable,
9718                   epcgAntRdPntEraseFailuresNotifLevel,
9719                   epcgAntRdPntLockOperations,
9720                   epcgAntRdPntLockFailures,
9721                   epcgAntRdPntLockFailuresNotifEnable,
9722                   epcgAntRdPntLockFailuresNotifLevel,
9723                   epcgAntRdPntPowerLevel,
9724                   epcgAntRdPntNoiseLevel,
9725                   epcgAntRdPntTimeEnergized,
9726                   epcgAntRdPntMemoryReadOperations
9727     }
9728     STATUS      current
9729     DESCRIPTION
9730         "Reader - Antenna Read Point objects."
9731     ::= { epcgReaderGroups 8 }
9732
9733
9734     epcgReaderAntennaSuppressNotifyGroup OBJECT-GROUP
9735     OBJECTS      { epcgAntRdPntReadFailureSuppressInterval,
9736                   epcgAntRdPntReadFailureSuppressions,
9737                   epcgAntRdPntWriteFailureSuppressInterval,
9738                   epcgAntRdPntWriteFailureSuppressions,
9739                   epcgAntRdPntKillFailureSuppressInterval,
9740                   epcgAntRdPntKillFailureSuppressions,
9741                   epcgAntRdPntEraseFailureSuppressInterval,
9742                   epcgAntRdPntEraseFailureSuppressions,
9743                   epcgAntRdPntLockFailureSuppressInterval,
9744                   epcgAntRdPntLockFailureSuppressions
9745     }
9746     STATUS      current
9747     DESCRIPTION
9748         "Reader - Antenna Read Point suppress notifications."
9749     ::= { epcgReaderGroups 9 }
9750

```

```

9751
9752 epcgReaderIOPortGroup OBJECT-GROUP
9753     OBJECTS     { epcgIoPortName,
9754                 epcgIoPortAdminStatus,
9755                 epcgIoPortOperStatus,
9756                 epcgIoPortOperStatusNotifEnable,
9757                 epcgIoPortOperStatusNotifFromState,
9758                 epcgIoPortOperStatusNotifToState,
9759                 epcgIoPortOperStatusNotifLevel,
9760                 epcgIoPortDescription,
9761                 epcgIoPortOperStatusPrior,
9762                 epcgIoPortOperStateSuppressInterval,
9763                 epcgIoPortOperStateSuppressions
9764             }
9765     STATUS      current
9766     DESCRIPTION
9767         "Reader IO Port objects."
9768     ::= { epcgReaderGroups 10 }
9769
9770 epcgReaderSourceGroup OBJECT-GROUP
9771     OBJECTS     { epcgSrcName,
9772                 epcgSrcReadCyclesPerTrigger,
9773                 epcgSrcReadDutyCycle,
9774                 epcgSrcReadTimeout,
9775                 epcgSrcGlimpsedTimeout,
9776                 epcgSrcObservedThreshold,
9777                 epcgSrcObservedTimeout,
9778                 epcgSrcLostTimeout,
9779                 epcgSrcUnknownToGlimpsedTrans,
9780                 epcgSrcGlimpsedToUnknownTrans,
9781                 epcgSrcGlimpsedToObservedTrans,
9782                 epcgSrcObservedToLostTrans,
9783                 epcgSrcLostToGlimpsedTrans,
9784                 epcgSrcLostToUnknownTrans,
9785                 epcgSrcAdminStatus,
9786                 epcgSrcOperStatus,
9787                 epcgSrcOperStatusNotifEnable,
9788                 epcgSrcOperStatusNotifFromState,
9789                 epcgSrcOperStatusNotifToState,
9790                 epcgSrcOperStatusNotifyLevel,
9791                 epcgSrcSupportsWriteOperations,
9792                 epcgSrcOperStatusPrior,
9793                 epcgSrcOperStateSuppressInterval,
9794                 epcgSrcOperStateSuppressions
9795             }
9796     STATUS      current
9797     DESCRIPTION
9798         "Reader (Tag Data) Source objects."
9799     ::= { epcgReaderGroups 11 }
9800
9801 epcgReaderNotifChannelGroup OBJECT-GROUP
9802     OBJECTS     { epcgNotifChanName,
9803                 epcgNotifChanAddressType,
9804                 epcgNotifChanAddress,
9805                 epcgNotifChanLastAttempt,
9806                 epcgNotifChanLastSuccess,
9807                 epcgNotifChanAdminStatus,
9808                 epcgNotifChanOperStatus,
9809                 epcgNotifChanOperNotifEnable,
9810                 epcgNotifChanOperNotifLevel,
9811                 epcgNotifChanOperNotifFromState,
9812                 epcgNotifChanOperNotifToState,

```

```

9813         epcgNotifChanOperStatusPrior,
9814         epcgNotifChanOperStateSuppressInterval,
9815         epcgNotifChanOperStateSuppressions
9816     }
9817     STATUS      current
9818     DESCRIPTION
9819         "Notification Channel objects"
9820     ::= { epcgReaderGroups 12 }
9821
9822     epcgReaderTriggerGroup OBJECT-GROUP
9823     OBJECTS      { epcgTrigName,
9824                   epcgTrigType,
9825                   epcgTrigParameters,
9826                   epcgTriggerMatches,
9827                   epcgTrigIoPort
9828                 }
9829     STATUS      current
9830     DESCRIPTION
9831         "Reader Trigger objects."
9832     ::= { epcgReaderGroups 13 }
9833
9834     epcgReaderTableAssocGroup OBJECT-GROUP
9835     OBJECTS      { epcgNotifTrigRowStatus,
9836                   epcgReadTrigRowStatus,
9837                   epcgNotifChanSrcRowStatus
9838                 }
9839     STATUS      current
9840     DESCRIPTION
9841         "Contains objects that maintain associations between rows
9842         in different tables."
9843     ::= { epcgReaderGroups 14 }
9844
9845     epcgReaderMandatoryTableAssocGroup OBJECT-GROUP
9846     OBJECTS      { epcgRdPntSrcRowStatus
9847                   }
9848     STATUS      current
9849     DESCRIPTION
9850         "Contains objects that maintain associations between rows
9851         in different tables."
9852     ::= { epcgReaderGroups 15 }
9853
9854     END
9855

```

9856 **11 Acknowledgements**

9857 Disclaimer

9858

9859 Whilst every effort has been made to ensure that this document and the  
 9860 information contained herein are correct, EPCglobal and any other party involved  
 9861 in the creation of the document hereby state that the document is provided on an  
 9862 “as is” basis without warranty, either expressed or implied, including but not  
 9863 limited to any warranty that the use of the information herein with not infringe any  
 9864 rights, of accuracy or fitness for purpose, and hereby disclaim any liability, direct  
 9865 or indirect, for damages or loss relating to the use of the document.

9866

9867 Below is a list of more active participants and contributors in the development of  
 9868 RM 1.0.1. This list does not acknowledge those who only monitored the process  
 9869 or those who chose not to have their name listed here. Active participants status  
 9870 was granted to those who generated emails, attended face-to-face meetings and  
 9871 conference calls that were associated with the development of this Standard.

9872

First Name	Last Name	Company	Notable Role
Martin	Jackson	Wal-Mart	Working Group Co-Chair
Ricardo	Labiaga	Formerly of Sun Microsystems	Working Group Co-Chair
Mark	Frey	EPCglobal Inc.	Working Group Facilitator
Bud	Biswas	Polaris Networks	
Gerhard	Gangl	7iD (formerly EOSS GmbH)	
Valentina	Shkolnikov	Alien Technology	
Ken	Jett	Cisco	
Craig	Sayers	Hewlett-Packard Co.	
Mark	Ulrich	PepsiCo	
Steven	Shafer	Microsoft	
Anush	Kumar	Microsoft	
Christian	Floerkemeier	Auto-ID Labs	
John	Gallant	Paxar	
Margaret	Wasserman	ThingMagic, LLC	
Lars-Erik	Helander	Intermec	
Debottam	Chatterjee	Polaris Networks	
Charan	Singh	Alien Technology	
Software Development Team		Impinj	

Jason	Yerardi	Paxar	
-------	---------	-------	--

9873

9874 The following list contains all companies that were opt'd-in to the Reader  
 9875 Management Working Group and have signed the EPCglobal IP Policy.

9876

Company
7iD (formerly EOSS GmbH)
Accenture
Acer Cybercenter Service Inc.
ACNielsen
ACSIS
Adtio Group Limited
Afilias Canada Corp
Albertsons
Alien Technology
Allixon
Altria Group, Inc./Kraft Foods
AMCO TEC International Inc.
Applied Wireless (AWID)
Ark Tech Ltd
AT4 Wireless (formerly Cetecom)
Auto-ID Labs - ADE
Auto-ID Labs - Cambridge
Auto-ID Labs - Fudan University
Auto-ID Labs - ICU
Auto-ID Labs - Japan
Auto-ID Labs - MIT
Auto-ID Labs - Univerisity of St Gallen
AXWAY/formerly Cyclone
BEA Systems
Beijing Futianda Technology Co. Ltd.
British Telecom
CAEN
Cap Gemini Ernst & Young
Certus Warensichenrung-Sys GmbH
Ceyon Technology Co., Ltd

Champtek
Cheng-Loong Corporation
China Elite Technology Co. Ltd
CISC Semiconductor
Cisco
Code Plus, Inc.
Computer Network Info Cntr.
Convergence Sys Ltd
Cybernette
D&S Technology
deister electronic GmbH
Denso Wave Inc
ECO, Inc.
Electronics and Telecommunication Research Institute (ETRI)
EPCglobal Inc.
FEIG Electronic
France Telecom
Fujitsu Ltd
Georgia-Pacific
GlobeRanger
GS1 Australia EAN
GS1 China
GS1 France
GS1 Germany (CCG)
GS1 Hong Kong
GS1 International
GS1 Japan
GS1 Netherlands (EAN.nl)
GS1 South Korea
GS1 Sweden AB (EAN)
GS1 Switzerland
GS1 Taiwan (EAN)
GS1 UK
GS1 US
Hewlett-Packard Co. (HP)

IBM
ID TechEx
Impinj
Indicus Software Pvt Ltd
Industrial Technology Research Institute
Infosys Technologies Limited
Infratab
Institute for Information Industry
Intelleflex
Intermec
Internet Initiative Japan, Inc.
Johnson & Johnson
Kim Hiap Lee Co.
Kimberly-Clark
KL-NET
Korea Computer Servs, Ltd
KTNET - KOREA TRADE NETWORK
LIT (Research Ctr for Logistics Info Tech)
LXE Inc.
Lyngsoe Systems
Lynko Technologies, Inc.
Manhattan Associates
McKesson
MET Labs
Metarights
Metro
Microelectronics Technology, Inc.
Microsoft
Mitsui
Mstar Semiconductor
National Computerization Agency
NCR
NEC Corporation
Nippon Telegraph & Telephone Corp (NTT)
NXP Semiconductors
OatSystems



Omnitrol Networks, Inc.
Oracle Corporation
Panda Logistics Co.Ltd
Pango Networks, Inc.
Patni Computer Systems
Paxar
PepsiCo
Polaris Networks
Pretide Technology, Inc.
Printronic
Procter & Gamble
PSC Scanning
Psion Teklogix Inc.
Q.E.D. Systems
Regal Scan Tech
RetailTech
Reva Systems
RFIT Solutions GmbH
Rush Tracking Systems
Samsung Electronics
Sanion Co Ltd
SAP
Savi Technology
Sedna Systems, Ltd.
SeeBeyond Technology (bought by Sun)
Shipcom Wireless, Inc.
Sirit
Skandsoft Technologies Pvt.Ltd.
Sun Microsystems
Supply Insight, Inc.
T3C Incorporated
TagSys
Tata Consultancy Services
ThingMagic, LLC
Tibco
Tongfang Microelectronics Co.Ltd.

Toppan Printing Co
Toray International, Inc.
TTA Telecommunications Technology Association
Tyco / ADT
Unisys
Ussen Limited Company
Venture Research
VeriSign
Vocollect
Vue Technology
Wal-Mart
Waldemar Winckel GmbH & Co. KG
Wish Unity (formerly Track-IT RFID)
WJ Communications
Yuen Foong Yu Paper

9877

9878 **12 References**

9879 [RP1] “EPCglobal Reader Protocol, Version 1.1,” EPCglobal,  
9880 [http://standards.epcglobalinc.org/RP\\_1\\_1](http://standards.epcglobalinc.org/RP_1_1)

9881 [MIBII] “Management Information Base for Network Management of TCP/IP-based  
9882 internets: MIB-II,<http://www.ietf.org/rfc/rfc1213.txt?number=1213>

9883 [ISODir2] ISO, “Rules for the structure and drafting of International Standards  
9884 (ISO/IEC Directives, Part2, 2001, 4<sup>th</sup> edition),” July 2002.

9885 [SYSLOG] “The syslog Protocol” IETF draft.  
9886 <http://www.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-syslog-protocol-15.txt>