

PacketCable™

Security, Monitoring, and Automation Specification

PKT-SP-SMA-I01-081121

ISSUED

Notice

This PacketCable specification is the result of a cooperative effort undertaken at the direction of Cable Television Laboratories, Inc. for the benefit of the cable industry and its customers. This document may contain references to other documents not owned or controlled by CableLabs. Use and understanding of this document may require access to such other documents. Designing, manufacturing, distributing, using, selling, or servicing products, or providing services, based on this document may require intellectual property licenses from third parties for technology referenced in this document.

Neither CableLabs nor any member company is responsible to any party for any liability of any nature whatsoever resulting from or arising out of use or reliance upon this document, or any document referenced herein. This document is furnished on an "AS IS" basis and neither CableLabs nor its members provides any representation or warranty, express or implied, regarding the accuracy, completeness, noninfringement, or fitness for a particular purpose of this document, or any document referenced herein.

© Copyright 2008 Cable Television Laboratories, Inc.
All rights reserved.

Document Status Sheet

Document Control Number:	PKT-SP-SMA-I01-081121			
Document Title:	Security, Monitoring, and Automation Specification			
Revision History:	I01 – Released 11/21/08			
Date:	November 21, 2008			
Status:	Work in Progress	Draft	Issued	Closed
Distribution Restrictions:	Author Only	CL/Member	CL/Member/Vendor	Public

Key to Document Status Codes

Work in Progress	An incomplete document, designed to guide discussion and generate feedback that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Issued	A stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Trademarks

CableLabs®, DOCSIS®, EuroDOCSIS™, eDOCSIS™, M-CMTS™, PacketCable™, EuroPacketCable™, PCMM™, CableHome®, CableOffice™, OpenCable™, OCAP™, CableCARD™, M-Card™, DCAS™, tru2way™, and CablePC™ are trademarks of Cable Television Laboratories, Inc.

Contents

1	SCOPE	1
1.1	Introduction and Purpose.....	1
1.2	Requirements.....	1
2	REFERENCES	2
2.1	Normative References.....	2
2.2	Informative References.....	3
2.3	Reference Acquisition.....	3
3	TERMS AND DEFINITIONS	4
4	ABBREVIATIONS AND ACRONYMS	5
5	OVERVIEW	6
5.1	PacketCable SMA.....	6
5.2	SMA Functional Entities.....	7
5.3	SMA Functional Areas.....	9
5.3.1	<i>Signaling</i>	9
5.3.2	<i>Media Session Establishment</i>	9
5.3.3	<i>QoS</i>	9
5.3.4	<i>Security</i>	10
6	PACKETCABLE SMA SIGNALING INTERFACE REQUIREMENTS	11
6.1	SMA Signaling.....	11
6.1.1	<i>HTTP Protocol Usage and URI Formation</i>	12
6.1.2	<i>SMA Messages</i>	13
6.1.3	<i>Registration and Deregistration</i>	15
6.1.4	<i>Heartbeat</i>	17
6.1.5	<i>Error Handling</i>	18
6.1.6	<i>Backoff and Retry Procedures</i>	19
6.1.7	<i>SMA Logic</i>	20
6.2	Media.....	20
6.2.1	<i>HTTP URI usage</i>	22
6.2.2	<i>Transport</i>	22
6.2.3	<i>Codecs</i>	22
6.2.4	<i>NAT and Firewall traversal</i>	22
6.3	Quality of Service.....	23
6.3.1	<i>QoS Usage</i>	24
6.3.2	<i>Procedures</i>	24
6.4	Security.....	25
6.4.1	<i>Authentication</i>	25
6.4.2	<i>Message Integrity and Protection</i>	26
6.4.3	<i>TLS Profile</i>	26
6.4.4	<i>TLS Certificate Profile and Validation</i>	26
ANNEX A	SMA DATA MODELS (NORMATIVE)	28
A.1	Instruction.....	28
A.2	Event.....	28
A.3	Registration.....	29
A.4	Deregistration.....	29
A.5	Response.....	30

A.6	Device List.....	30
A.7	Heartbeat.....	31
A.8	Logic.....	31
A.9	Device Profile.....	35
ANNEX B SMA LOGIC (NORMATIVE).....		46
B.1	Macros and Statements.....	46
B.2	Assignments.....	46
B.3	Method/Macro Invocation.....	47
B.4	Wait Statement.....	47
B.5	Local Variable Usage.....	47
B.6	Expressions.....	47
B.7	Conditionals.....	48
B.8	If-Else.....	48
B.9	While.....	49
B.10	Until.....	49
B.11	Switch.....	49
B.12	Temporal Expressions.....	49
B.13	Return Values From a Macro.....	50
B.14	Triggers.....	51
B.15	Schedules.....	51
APPENDIX I SMA SIGNALING EXAMPLES.....		52
I.1	Retrieve a specific property of an SMA device.....	52
I.2	To retrieve all properties specific to an SMA device.....	53
I.3	Retrieving a specific set of properties for an SMA device.....	53
I.4	To retrieve all properties of all SMA devices connected to an SMA gateway.....	54
I.5	Retrieve an SMA gateway's devices, capabilities and macros.....	55
I.6	Updating a Property.....	56
APPENDIX II SMA LOGIC EXAMPLES.....		57
II.1	Example 1.....	57
II.1.1	Example 1a.....	57
II.1.2	Example 1b.....	57
II.2	Example 2.....	57
II.3	Example 3.....	57
II.4	Example 4.....	58
II.5	Example 5.....	58
APPENDIX III SMA DEVICE PROFILE USAGE.....		60
III.1	Example Light Switch XML Schema.....	60
III.2	Example Device Listing.....	61
APPENDIX IV ACKNOWLEDGEMENTS.....		62

Figures

Figure 1 - SMA Reference Architecture.....	6
Figure 2 - SMA Entities and Interactions.....	8
Figure 3 - SMA Interfaces.....	11
Figure 4 - SMA Signaling Interface.....	11
Figure 5 - SMA Media Session Establishment.....	21
Figure 6 - Interfaces to support media in the presence of NATs.....	23
Figure 7 - SMA QoS Interfaces.....	23

Tables

Table 1 - Event Server behavior upon receiving HTTP responses from the SMA gateway.....	18
Table 2 - SMA gateway behavior upon receiving HTTP responses from the Event Server.....	18
Table 3 - QoS Interface Descriptions.....	24
Table 4 - TLS Certificate Profile.....	26

This page is left blank intentionally.

1 SCOPE

1.1 Introduction and Purpose

PacketCable SMA is a CableLabs effort designed to address the following areas.

- **Security** - The ability to protect lives and property through professional monitoring, notifying first responders (e.g., fire, police, and medical) and providing emergency alerts to authorized individuals like home occupants. For example, fire and smoke sensors alert monitoring agencies who can notify and provide critical information to local emergency personnel.
- **Monitoring** - The ability to self-monitor the status and activity in the home so that a user can be made aware of any desired state changes. For example, when motion is detected by a motion sensor, real-time alerts such as video or photo clips are transmitted to the desired recipients.
- **Automation** - The ability to automate and remotely control lifestyle conveniences such as lighting, heating, cooling and appliances. For example, a user can remotely use a web portal to verify and control conditions such as lighting or temperature in the home.

This document specifies the Signaling, Media, Quality of Service (QoS), and Security requirements to support the PacketCable SMA Architecture. For the SMA gateway device architecture, provisioning and management requirements please refer to [PKT-SMA-PROV].

1.2 Requirements

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST"	This word means that the item is an absolute requirement of this specification.
"MUST NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
"MAY"	This word means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

2 REFERENCES

2.1 Normative References

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Notwithstanding, intellectual property rights may be required to use or implement such normative references.

- [ID ICE] IETF Internet-Draft, draft-ietf-mmusic-ice-19, Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, October 29, 2007, work in progress.
- [ID TURN] IETF Internet-Draft, draft-ietf-behave-turn-11, Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN), October 29, 2008, work in progress.
- [PKT-CODEC-MEDIA] PacketCable 2.0 Codec and Media Specification, PKT-SP-CODEC-MEDIA-I05-080710, July 10, 2008, Cable Television Laboratories, Inc.
- [PKT-MM] PacketCable Multimedia Specification, PKT-SP-MM-I04-080522, May 22, 2008, Cable Television Laboratories, Inc.
- [PKT-MM-WS] PacketCable Multimedia Web Service Interface Specification, PKT-SP-MM-WS-I02-080522, May 22, 2008, Cable Television Laboratories, Inc.
- [PKT-SEC1.5] PacketCable 1.5 Security Specification, PKT-SP-SEC1.5-I02-070412, April 12, 2007, Cable Television Laboratories, Inc.
- [PKT-SMA-PROV] PacketCable SMA Provisioning Specification, PKT-SP-SMA-PROV-I01-081121, November 21, 2008, Cable Television Laboratories, Inc.
- [RFC 2616] IETF RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1, June 1999.
- [RFC 2617] IETF RFC 2617, HTTP Authentication: Basic and Digest Access Authentication, June 1999.
- [RFC 2818] IETF RFC 2818, HTTP over TLS, May 2000.
- [RFC 3264] IETF RFC 3264, An Offer/Answer Model with the Session Description Protocol (SDP), June 2002.
- [RFC 3268] IETF RFC 3268, Advanced Encryption Standard (AES) cipher suites for Transport Layer Security (TLS), June 2002.
- [RFC 4346] IETF RFC 4346, The Transport Layer Security (TLS) Protocol Version 1.1, April 2006.
- [RFC 4566] IETF RFC 4566, SDP: Session Description Protocol, July 2006.
- [RFC 5280] IETF RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008.
- [RFC 5389] IETF RFC 5389, Session Traversal Utilities for NAT (STUN), October 2008.
- [W3 XML1.0] Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation, August 2006.
- [W3 XSD1.0] XML Schema Part 1: Structures Second Edition, W3C Recommendation 28, October 2004.

2.2 Informative References

This specification uses the following informative references.

- [ID-ICE-NONSIP] IETF Internet-Draft, draft-rosenberg-mmusic-ice-nonsip-01, "Guidelines for Usage of Interactive Connectivity Establishment (ICE) by non Session Initiation Protocol (SIP) Protocols", July 14, 2008, work in progress.
- [TR-MM-ARCH] PacketCable Multimedia Architecture Framework Technical Report, PKT-TR-MM-ARCH-V02-051221, December 21, 2005, Cable Television Laboratories, Inc.
- [TR-PKT-NFT] PacketCable NAT and Firewall Traversal Technical Report, PKT-TR-NFT-V05-080425, April 25, 2008, Cable Television Laboratories, Inc.
- [TR-SMA-ARCH] PacketCable SMA Architecture Framework Technical Report, PKT-TR-SMA-ARCH-V01-081121, November 21, 2008, Cable Television Laboratories, Inc.

2.3 Reference Acquisition

- Cable Television Laboratories, Inc., 858 Coal Creek Circle, Louisville, CO 80027; Phone +1-303-661-9100; Fax +1-303-661-9199; <http://www.cablelabs.com>
- Internet Engineering Task Force (IETF), Internet: <http://www.ietf.org/>
Note: Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time.
The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.
Internet-Drafts may also be accessed at <http://tools.ietf.org/html/>
- World Wide Web Consortium, www.w3c.org, c/o MIT, 32 Vassar Street, Room 32-G515, Cambridge, MA 0213.

3 TERMS AND DEFINITIONS

This specification uses the following terms:

Conditional	An XML element that provides for conditional execution of statements
DSx Signaling	DSA, DSC, and DSD – the full range of Dynamic Service messaging
Event	A change in a defined condition
Expression	An XML element that resolves to a value
Macro	A series of steps and logic that can be run
NAT, NAPT	NATs perform IP address translation, typically interconnecting private and public address domains. NAPT devices also translate ports in order to save IP addresses. In this document, the term NAT also refers to NAPT devices.
PacketCable Multimedia	An application agnostic QoS architecture for services delivered over DOCSIS networks
RESTful web services	Use of REST design principles via HTTP as the protocol
Schedule	A time-based event generator
Statement	A single step in a macro
Trigger	Something that responds to an event by running a program

4 ABBREVIATIONS AND ACRONYMS

This specification uses the following abbreviations:

AS	Application Server
AM	Application Manager
CM	Cable Modem
CMTS	Cable Modem Termination System
DOCSIS	Data-Over-Cable Service Interface Specifications
HTTP	Hyper Text Transport Protocol
ICE	Interactive Connectivity Establishment
IP	Internet Protocol
MPEG	Moving Picture Experts Group
NAT	Network Address Translation
PCMM	PacketCable Multimedia
PS	Policy Server
QoS	Quality of Service
REST	REpresentational State Transfer
SMA	Security, Monitoring and Automation
SMS	Short Message Service
STUN	Simple Traversal of UDP Through NAT
TLS	Transport Layer Security
TURN	Traversal Using Relay NAT
XML	eXtensible Markup Language

5 OVERVIEW

The PacketCable Security, Monitoring and Automation (SMA) architecture describes a set of functional groups and logical entities, as well as a set of interfaces that support the information flows exchanged between the entities.

This section provides:

- An overview of PacketCable SMA.
- An overview of the functional entities and areas addressed by this specification.

The remainder of this section provides an brief overview of PacketCable SMA, the functional area and entities. Please refer to [TR-SMA-ARCH] for a detailed overview of PacketCable SMA.

5.1 PacketCable SMA

PacketCable SMA, as specified in this document identifies a set of functional groups and logical entities as illustrated in Figure 1.

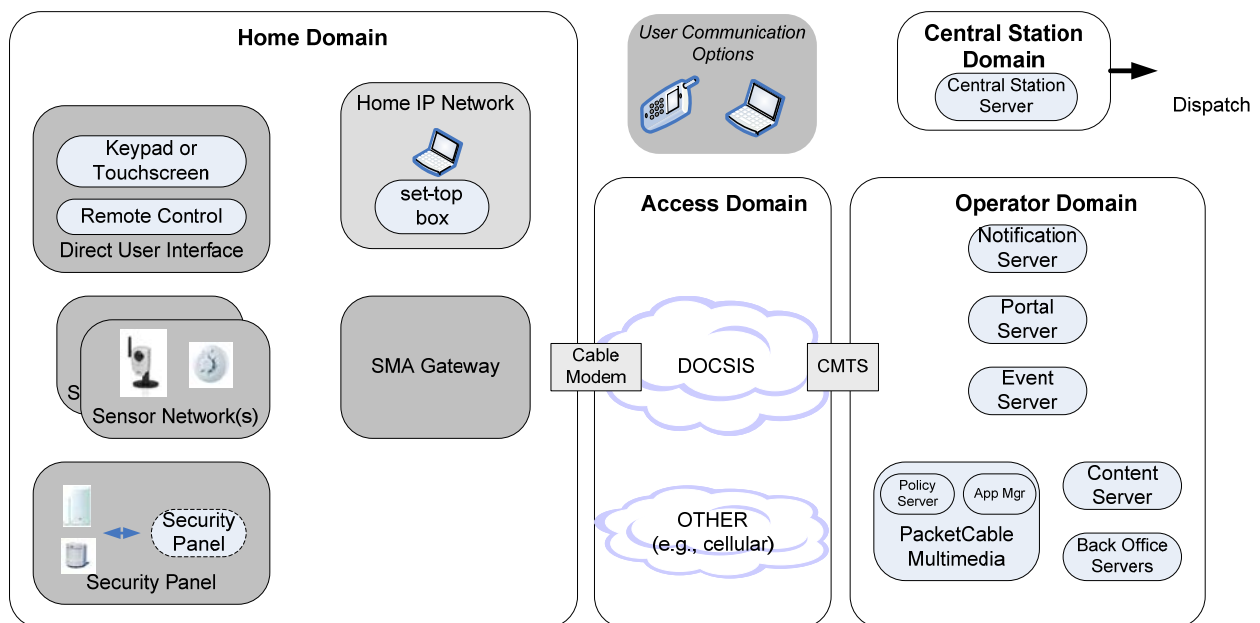


Figure 1 - SMA Reference Architecture

There are four domains that are identified, as described below:

- **Home Domain:** The Home Domain consists of the SMA gateway which communicates with servers in the Operator Domain as well as entities in the home such as legacy security panels, sensors and controller networks, home Internet protocol (IP) networks, and user interfaces (keypads, remote controls, Set-top Box OCAP application, etc.).
- **Access Domain:** The Access Domain provides the communications path and QOS resources required to facilitate communications between the SMA gateway and the Operator Domain. The Access Domain includes the Cable Modem, and optional router, in the home as well as the Cable Modem Termination System in the network.

- **Operator Domain:** The Operator Domain contains an Event Server which communicates with the SMA gateway. It also contains a Portal Server which allows the end user to check status, arm/disarm, and manage the system through various interfaces such as web user interface. Other elements in the Operator Domain include Operational Support Systems to facilitate billing, provisioning, customer care, and network management.
- **Central Station Domain:** The Central Station Domain contains hardware, software and communications facilities required to receive critical alarms from the Event Server.

5.2 SMA Functional Entities

Figure 2 depicts typical SMA entities and their relationships, as relevant to the PacketCable SMA efforts. For simplicity, the diagram indicates two domains, Operator and Home. Each domain contains a set of logical or physical components.

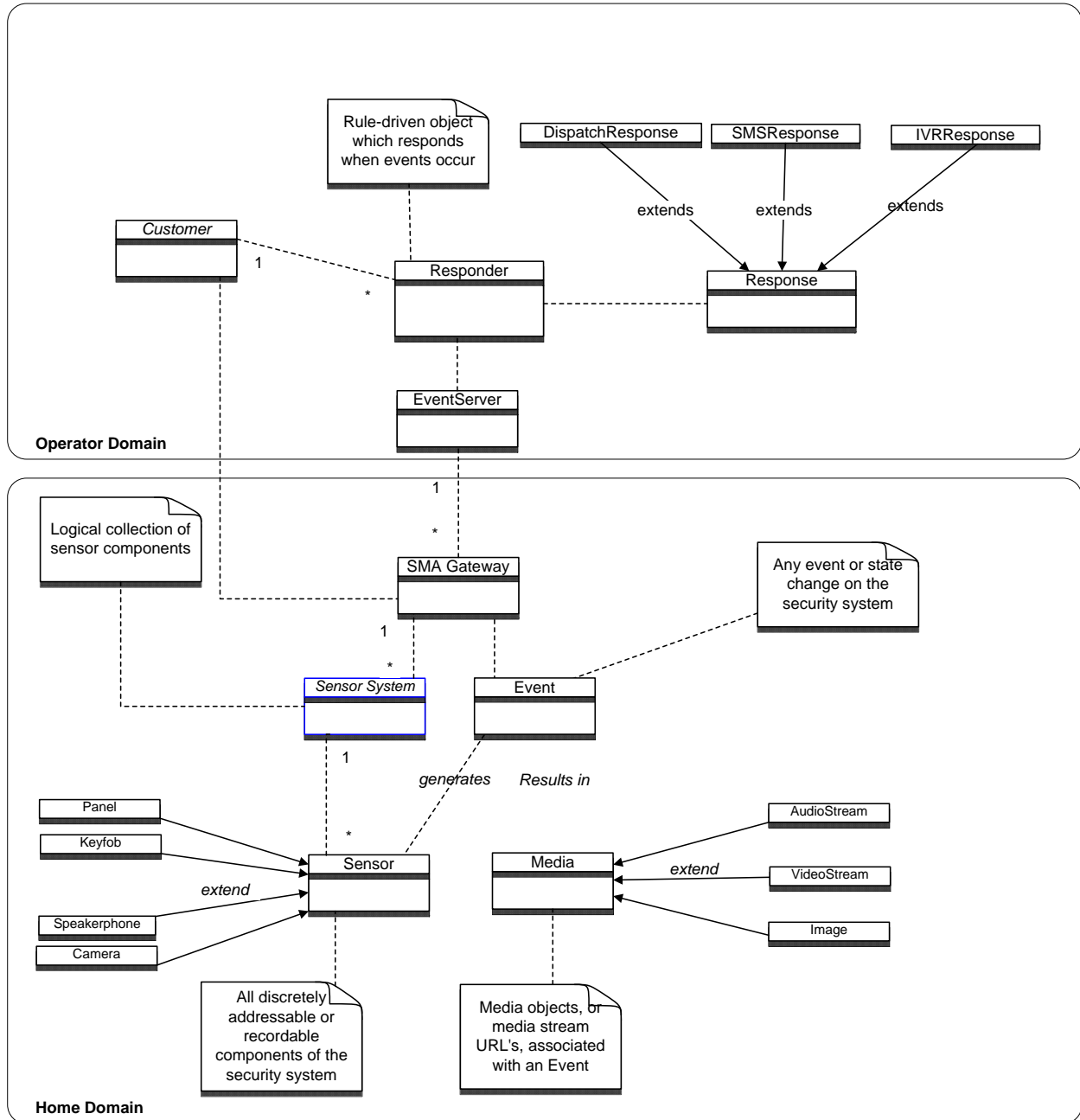


Figure 2 - SMA Entities and Interactions

PacketCable SMA specified in this document focuses on the interface between two primary entities: the SMA gateway and the Event Server. The SMA gateway resides in the customer premises, and acts as the conduit to signaling interactions between the Operator and the SMA devices in the home. Communication from the Operator domain is accomplished using the Event Server, the SMA gateway's counterpart in the MSO domain.

In the customer premise, the SMA gateway is responsible for communications with one or more security systems ("System"). Each System is a logical collection of one or more Components in the security system. Each Component, in turn, represents Controls, Sensors, or both. Examples of Sensors include magnets (for doors and windows), motion sensors, speakerphones (for listening in), and video cameras. These sensors are primarily unidirectional, reporting state or other sensory information to the Gateway, but not accepting state changes.

Controls are similar to Sensors in that they represent devices deployed throughout the home. Controls are also controllable, accepting requests for state changes from the SMA gateway. Examples of Controls include Lighting controls, and the security Panel itself (arming and disarming it).

When a Security System detects a state change (e.g., in a sensor), it reports a security Event, a data object which identifies the sensor and the state change. Each Event may be associated with one or more media data objects, to include rich media information which is contextual to the event. Media types can be audio streams, video streams, or still images.

The Event Server receives all events from the SMA gateway. When an Event is received, the Event Server will iterate through one or more processing rules, termed Responders, which are rule-driven objects that listen for Events of certain kinds, and respond appropriately. For example, one Responder might watch for Burglary events, and respond with "dispatch the Police". Another Responder might watch for Arming or Disarming events, and respond with "send an SMS message to the customer". Each of these types of responses (dispatch, SMS, etc.) are represented by Response objects, for which three specific sample types are shown: Dispatch Response, SMS Response, and IVR Response (IVR stands for "Interactive Voice Response", also known as "automated outbound phone call").

5.3 SMA Functional Areas

PacketCable SMA, as specified in this document, addresses four functional areas: Signaling, Media, QoS, and Security. These areas are described in the following sub-sections.

5.3.1 Signaling

PacketCable SMA Signaling refers to the communication between the SMA gateway and the Event Server. This communication interface allows an MSO to control an SMA gateway and for the SMA gateway to respond to queries, and report required events.

To be able to communicate, the SMA gateway and the Event Server need to be aware of each other. The SMA gateway then needs to initiate communication with the Event Server and register to indicate its availability. Once registered, the SMA gateway can communicate any events generated by the SMA devices in the Home Domain. This also allows an Event Server to communicate MSO directives and information requests to the SMA gateway.

This interface is based on RESTful web services.

5.3.2 Media Session Establishment

Media in SMA may include both video and audio streams, also referred to as media sessions. Video sessions would generally be one way streaming video, although bidirectional video is permissible. Audio sessions may be either one-way or two-way. An example of a two-way audio stream would be a user pressing a panic button on an alarm panel and establishing a call to the central station.

Media session establishment uses the Session Description Protocol (SDP) and the Offer/Answer model, as specified in [RFC 4566] and [RFC 3264], respectively.

5.3.3 QoS

The Quality of Service for SMA is based on PacketCable Multimedia defined in [TR-MM-ARCH]. The components that participate in QoS management in SMA take on the same behavior and have the same responsibilities as defined in the PCMM specifications. Please refer to [TR-MM-ARCH] for more information on how the CM, CMTS, and Policy Server components participate in QoS management.

The Event Server is responsible for identifying SMA requests that require QoS protection for packet flows that traverse the DOCSIS access network. The Event Server is also responsible for reserving, committing, and releasing QoS resources for all such flows.

The Application Manager serves as the interface between the PacketCable SMA architecture and the PacketCable Multimedia architecture. Its function is to receive QoS requests from the Event Server and to create and manage the PacketCable Multimedia Gates for each flow using the COPS-based pkt-mm-3 interface.

The communication between the Event Server and the Application Manager is based on the PacketCable Multimedia Web Service Interface Specification defined in [PKT-MM-WS]. The choice of a web service for this interface was made to allow for code reuse within the Event Server, which already has a RESTful web services interface for communication with the SMA gateway.

5.3.4 Security

SMA signaling contains potentially sensitive and personally identifiable information, such as status change events indicating conditions inside the house and instructions controlling SMA devices, such as alarm settings. Exposing the SMA signaling interface to unauthorized individuals may result in an unacceptable security risk. Specifically, without security, threats such as false event insertion, rogue instruction insertion, private data acquisition, and other common man-in-the-middle, denial of service, and replay attacks may be possible.

To mitigate such security threats, it is recommended that the SMA signaling be protected by a security mechanism to ensure the privacy of the communications as well as the authenticity and integrity of the data transmitted over the communications channel. While DOCSIS provides BPI+ as a mechanism for ensuring message privacy, authentication, and integrity, there are some situations in which relying on BPI+ may not be sufficient to address the security risks. For example, deployments which have turned off BPI+ security mechanism and deployments in which the SMA gateway resides on a customer's local area network which is not protected by BPI+, another security mechanism may be needed.

In keeping with the restful approach for SMA signaling, security in SMA is accomplished using Transport Layer Security (TLS), as specified in [RFC 4346] and [RFC 3268]. The usage of TLS for message encryption and message integrity is governed by configuration data. An SMA gateway may be configured to use TLS for all communication with its Event Server. The Event Server may be configured to only accept communication with a particular SMA gateway when that communication is protected by TLS. Some SMA gateways communicating with an Event Server may be required to use TLS while other SMA gateways communicating with the same Event Server may not be required to use TLS.

6 PACKETCABLE SMA SIGNALING INTERFACE REQUIREMENTS

PacketCable SMA focuses primarily on the interactions between the SMA gateway and the Event Server, and any interfaces to support this communication. This specification describes the areas of Signaling, Media, QoS, and Security. The interfaces required to support this are illustrated in Figure 3.

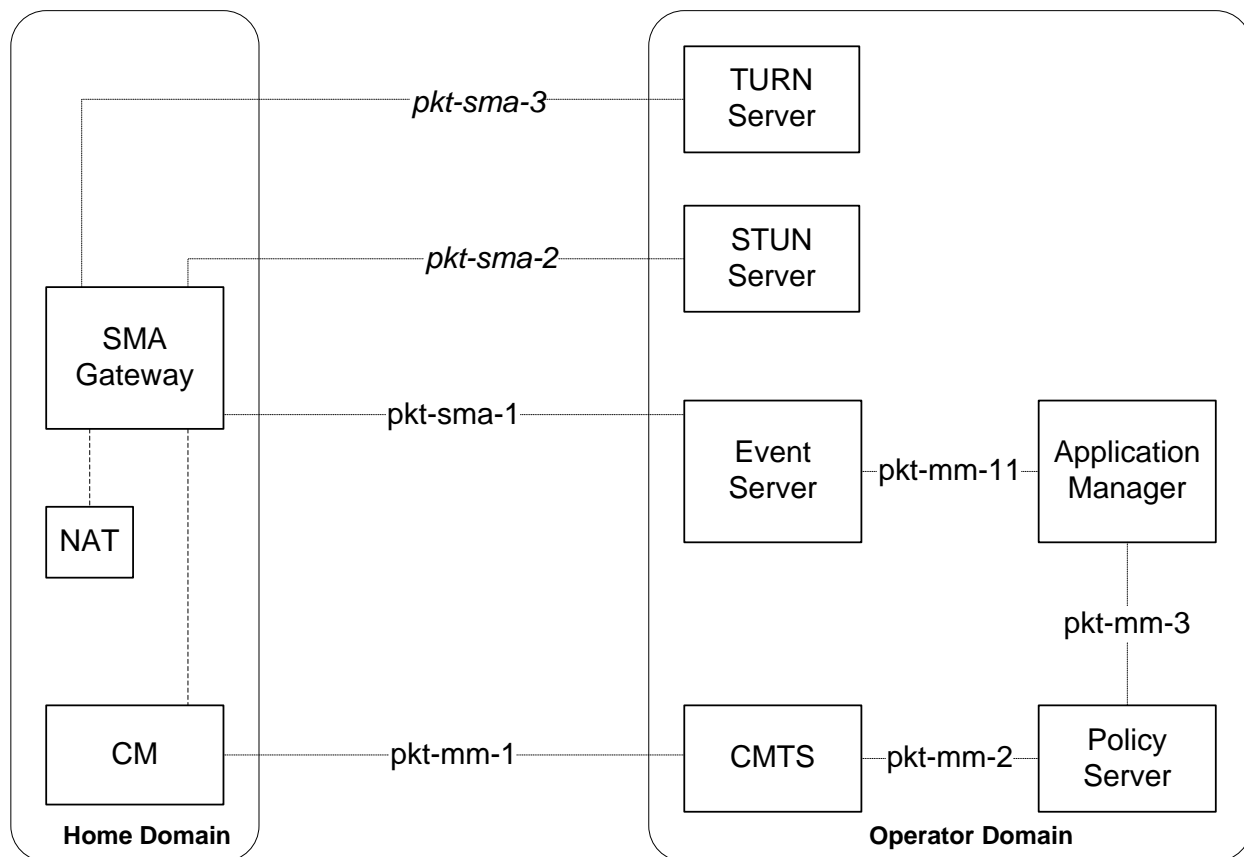


Figure 3 - SMA Interfaces

As shown in Figure 3, when the SMA gateway is connected via a DOCSIS Cable Modem, PacketCable SMA reuses the PacketCable Multimedia interfaces for QoS. It also specifies three interfaces: *pkt-sma-1*, *pkt-sma-2* and *pkt-sma-3*. The latter two are only required when the SMA gateway is connected via a NAT device, and hence are shown in italics. The functional areas addressed by these interfaces, and the interface requirements, are presented in the following sub-sections.

6.1 SMA Signaling

The SMA Signaling Interface is illustrated in Figure 4.

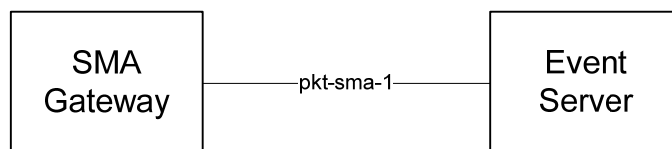


Figure 4 - SMA Signaling Interface

The interface, termed pkt-sma-1, allows for communication between the SMA gateway and the Event Server. The interface is realized via RESTful web services, which uses:

1. HTTP 1.1 ([RFC 2616]) as the communication protocol.
2. W3C XML 1.0 ([W3 XML1.0]) for data definitions, with the data models created using the W3C XML Schema Definition 1.0 ([W3 XSD1.0]).

This interface allows for the SMA gateway and the Event Server to send and receive SMA Messages. The requirements related to the HTTP protocol usage and URI formation, the SMA messages, error handling, and the backoff and retry mechanisms are specified in the following sub-sections.

6.1.1 HTTP Protocol Usage and URI Formation

This sub-section specifies the HTTP protocol usage and the HTTP URI formation. The following requirements apply:

- An SMA gateway implementing the pkt-sma-1 interface specified in this document **MUST** comply with the requirements specified in this sub-section.
- An Event Server implementing the pkt-sma-1 interface specified in this document **MUST** comply with the requirements specified in this sub-section.

The version of HTTP to be used for pkt-sma-1 **MUST** be HTTP 1.1, as specified in [RFC 2616]. The subset of possible HTTP operations that can be used for SMA messaging is limited to the following:

- GET
- POST
- PUT
- DELETE

The SMA gateway and the Event Server act as either the client or the server, based on the originator of an SMA message. When the SMA gateway is the originator of a message, it **MUST** use the following format for HTTP URI formation:

```
http://<Event Server FQDN>/<resource ID>[(?<query string>)]
```

When the Event Server is the originator of a message, it **MUST** use the following format for HTTP URI formation:

```
http://<SMA gateway FQDN>/<resource ID>/[(?<query string>)]
```

Where:

<Event Server FQDN > is the Event Server FQDN, provided during configuration.

<SMA gateway FQDN > is the identifier of the SMA gateway.

<resource ID> is the identifier for the resource.

<query string> additional filter criteria for a message type of 'instruction'.

The SMA gateway identifier (FQDN or URI) indicated above **SHOULD** include the Event Server Identifier. For example, if the Event Server FQDN is es.example.com, an SMA gateway that is connected to it should have an

FQDN of sg.es.example.com. This will allow network elements that wish to establish communication with an SMA gateway (sg.es.example.com) to easily identify the Event Server through which a communication channel can be established (es.example.com).

Certain resource identifiers are reserved by this document. They are as follows:

- ALLKNOWNMADEVICES, refers to all the SMA devices known to an SMA gateway.
- ALLACTIVESMADEVICES, refers to the subset of all known SMA devices that are actively connected to an SMA gateway.
- ALLINACTIVESMADEVICES, refers to the subset of all known SMA devices that are not actively connected to an SMA gateway.
- <SMA Dev>/PROFILE, refers to the SMA device profile of the SMA device identified by <SMA Dev>.
- <SMA Dev>, refers to the current state of all the properties of the SMA device identified by <SMA Dev>.
- ev/<correlation ID | event ID, or HB>, refers to an event being generated, or the heartbeat.

When the SMA gateway related identifiers that refer to its connected SMA devices (above) are included in an instruction, the SMA gateway MUST return the applicable device identifiers, display names, namespaces, device tag (optional), schema locations (if the namespace is not a URI, or the URI does not indicate the Schema location) using the XML Schema specified in Annex A.6.

For the SMA device profile, refer to Annex A.9 and the example in Appendix III. This does not return the current state of the all properties. It returns the methods and properties supported by the specific device and related information such as property types, default values etc. When the SMA device is asked to return the current state of all the properties, the SMA device MUST return an SMA response (Annex A.5) that contains all the supported properties and their corresponding values.

6.1.2 SMA Messages

The messages exchanged between the SMA gateway and the Event Server are termed as SMA messages. This section provides more details regarding the SMA message types, message transmission, and their formats.

6.1.2.1 Message Types

SMA messages can be categorized into two types: Instructions and Events. The Event Server can send across messages that control the state of the SMA services provided by the SMA gateway and the SMA network elements in the Home Domain. The SMA gateway can also send across a request to request the property on an Event Server (e.g., its registration state). Such SMA messages are termed instructions. A non-exhaustive list of instruction messages follows:

- Add SMA device: adds a new SMA device to the SMA gateway.
- Delete SMA device: removes a SMA device from the SMA gateway.
- Set Property: sets a SMA device property.
- Get Property: gets a SMA device property.
- Query Home Domain: returns the queried properties (e.g., connection status) for SMA devices that are connected to the SMA gateway in the home domain.
- Create Logical Directive: a new directive (e.g., macros, triggers, schedules) is added.
- Modify Logical Directive: a previously configured logical directive is updated.
- Delete Logical Directive: removes a previously configured directive.

Messages can also be Events, which are messages that report the state of a particular entity or action on the originating entity. Events can be generated as a result of modification to SMA features, but they do not themselves modify the SMA features. They are transmitted from the SMA gateway to the Event Server and, optionally, the other way around. Examples of events include:

- SMA device connectivity, e.g., new SMA device found.
- SMA device state change: e.g., the motion sensor was triggered.

6.1.2.2 Message Transmission

Based on the RESTful web services approach for SMA signaling, each SMA message is transmitted via a HTTP method (e.g., HTTP GET). The HTTP method identifies the intent:

- The HTTP GET operation is used to transmit an instruction to retrieve a resource. A resource can be one or more properties, or property fields, within an Event Server, SMA gateway, or an SMA device. For example, the instruction can request a list of all SMA devices connected to an SMA gateway.
- The HTTP POST operation is used to identify an SMA Message instruction or an event. An example of an instruction is media session establishment. An example of an event is triggering of a sensor.
- The HTTP PUT operation is used to identify an instruction to modify the contents of a resource. For instance, the Event Server can instruct the SMA gateway to modify the contents of a specific resource or property. It is to be noted that such a resource already exist. For creation, a HTTP POST operation is used.
- The HTTP DELETE operation identifies an instruction to delete a specific resource. For example, the SMA gateway may be instructed to delete a connected SMA device.

The following requirements apply to the SMA gateway:

- The SMA gateway **MUST** support the HTTP methods - GET, POST, PUT, DELETE - as specified in this document. As a note, while the SMA gateway can receive a HTTP DELETE request, the SMA gateway **MUST NOT** use the HTTP DELETE operation when it communicates with the Event Server.
- Upon receiving a HTTP request from the SMA gateway, the Event Server **MUST** respond immediately with a suitable HTTP response (e.g., 200 OK). It is to be noted that if the HTTP method contains an instruction, the HTTP response may or may not contain an SMA response as specified later in this section.

The following requirements apply to the Event Server:

- The Event Server **MUST** support the HTTP methods - GET, POST, PUT and DELETE - as specified in this document.
- An Event Server that receives a HTTP DELETE operation from the SMA gateway **MUST** reject the request with a HTTP 403 response.
- Upon receiving a HTTP request from the SMA gateway, the Event Server **MUST** respond immediately with a suitable HTTP response (e.g., 200 OK). It is to be noted that when the HTTP method contains an instruction, the HTTP response may or may not contain an SMA response as specified later in this section.

When the HTTP request contains an instruction, the HTTP response may or may not contain an SMA response to the instruction. If the instruction resulted in the retrieval or generation of data, then there are two ways in which this data can be transmitted:

- Synchronous, i.e., the data is transmitted within the HTTP response.
- Asynchronous, i.e., the data is transmitted later as an event (within an SMA event).

The decision of synchronous or asynchronous is made based on the two timers: T1 and T2. Timer T1 indicates the acceptable time for a HTTP Response, with or without an SMA response to the SMA instruction. Timer T2 indicates the acceptable time for the SMA response. The following SMA gateway requirements apply:

- An SMA gateway that receives an instruction via a HTTP request **MUST** send an HTTP response within Timer T1.
- An SMA gateway **MUST** respond with an SMA message to an instruction within Timer T2.
- An Event Server that does not receive a HTTP response or an SMA response within the specified timers, T1 and T2 respectively, **MUST** declare the instruction to have failed and retry.

The following Event Server requirements apply:

- An Event Server that receives an instruction via a HTTP request **MUST** send an HTTP response within Timer T1.
- An Event Server **MUST** respond with an SMA response to an instruction within Timer T2.
- An SMA gateway that does not receive a HTTP response or an SMA response within the specified timers, T1 and T2 respectively, **MUST** declare the instruction to have failed and retry.

For asynchronous messages, the SMA requests and responses are correlated using a correlation ID. This correlation ID **MUST** be used as the event ID when sending an asynchronous event. For all asynchronous SMA responses, the cause of the event **MUST** be the label "instruction".

6.1.2.3 Message Format

The XML Schemas for the data inserted in the various SMA messages are specified in Annex A. The SMA gateway **MUST** adhere to the following XML Schema definitions:

- Instruction
- Event
- Registration
- Deregistration
- Response
- Device List
- Heartbeat

The Event Server **MUST** also adhere to the above XML Schema definitions. The XML Schemas and XML instance documents comply with [W3 XML1.0] and [W3 XSD1.0], respectively.

The Device Profile data model is provided for SMA device vendors in an effort to provide a common, interoperable, way in which a SMA device vendor can describe SMA device properties and methods. An example usage is illustrated in Appendix III.

6.1.3 Registration and Deregistration

This section specifies the registration and deregistration procedures for SMA services. They are specified in the following sub-sections.

6.1.3.1 Registration

The SMA gateway **MUST** register with the Event Server prior to exchanging SMA instructions or events. The registration process authenticates the SMA gateway with the Event Server and also allows the Event Server to bind an SMA gateway identity with a TCP or TLS connection terminating at the Event Server.

When TLS is used mutual authentication between the SMA gateway and the Event Server is accomplished via one of two mechanisms: mutual authentication using X.509 certificates or HTTP digest authentication ([RFC 2617]). When TLS is not used, digest authentication **MAY** be used. However, unless the TCP connection is integrity-protected using other mechanisms, it is subject to security threats such as Man-in-the-Middle and impersonation.

Thus, unless it is explicitly configured to not use TLS, the SMA Gateway **MUST** attempt to establish TLS with the Event Server. An SMA gateway that contains an X.509 certificate and is not explicitly configured to use digest (via the configuration of a username and password) **MUST** present its certificate during TLS establishment.

If the SMA gateway is configured with a username it will anticipate digest authentication and **MAY** include an Authorization header in the first SMA register request sent to the Event Server, as described in [RFC 2617]. If the SMA gateway includes an Authorization header field in the first SMA register request, the Authorization header field **MUST** contain an empty response parameter.

Whenever it attempts to establish a TLS connection, the SMA gateway **MUST** validate the Event Server certificate during TLS setup as specified in Section 6.4.4.2 of this document. The SMA gateway **MUST** tear down the TCP connection used for TLS communication if the server certificate is not valid.

Upon receiving the first register request from the SMA gateway, unless TLS was established via mutual authentication, the Event Server **SHOULD** challenge the register request with a 401 response. When challenging a register request, the Event Server **MUST** include a WWW-Authenticate message within the 401 response, as described in [RFC 2617]. In some cases, the Event Server **MAY** accept the SMA gateway's registration without challenge and answer the register request with a 200 response. The determination of when to challenge the SMA gateway's registration request and when to allow registration without challenge is outside the scope of this document. An example of where this behavior may be desired is in a situation where critical events need to be exchanged and authentication, privacy, and integrity are ensured by non-SMA means.

Upon receiving the 401 to the first register request, the SMA gateway **MUST** send a subsequent register request to the Event Server. The SMA gateway's second register request **MUST** include an appropriate response parameter calculated from the nonce challenge presented in the 401 response from the Event Server.

Upon receiving the subsequent register request from the SMA gateway, the Event Server **MUST** validate the response as per [RFC 2617], including validating that the nonce is not stale and that the response has been correctly computed based on the shared secret and the nonce in the 401 from the Event Server. If the Event Server finds that the response is invalid, it **MAY** respond with another 401 with a new nonce. Alternatively, if the Event Server finds that the response is invalid, it **MAY** drop the TCP or TLS connection completely.

Unless configured not to use TLS, the SMA gateway **MUST NOT** start the SMA registration procedure until a mutually authentication TLS connection is established. Once a TCP or TLS connection is established by the SMA gateway, the SMA gateway **MUST** send an SMA register message inside an HTTP POST request to the Event Server as an SMA event with a register body that is compliant with the XML Schema specified in A.3.

The SMA Gateway **MUST** use the following URI for a registration event:

`http://<Event Server FQDN>/ev/register`

If the Event Server finds that the response is valid, it **MUST** send a 200 response to the HTTP POST request. Once registration is completed the same channel (TCP or TLS) is used for bidirectional communication between the SMA gateway and the event server.

6.1.3.2 *Deregistration*

Deregistration can be initiated by either the SMA gateway or the Event Server. Deregistration may occur for various reasons. The SMA gateway may wish to deregister when it knows it needs to go offline for a period of time for a reboot, for example, thus gracefully closing the connection with the Event Server. The Event Server may wish to deregister when it needs to go offline for maintenance and thus needs to move existing registrations to another Event Server.

When it wishes to deregister, the SMA gateway **MUST** send a deregister message inside an HTTP POST request to the Event Server using the following URI:

`http://<Event Server FQDN>/ev/deregister.`

When the Event Server wishes to deregister the SMA gateway, the Event Server **MUST** send a deregister message inside an HTTP POST request to the SMA gateway using the following URI:

`http://<SMA gateway FQDN>/ev/deregister.`

The body of the deregister request in either case will contain a reason for deregistration (e.g., "user initiated reset", "load balancing").

6.1.3.3 *Body of register request*

The SMA gateway **MUST** include the following in the body of the register request:

- SMA gateway URI
- Manufacturer
- Model
- Version
- A comma separated list of SMA protocol versions that it supports (e.g., "1.0,1.1")

6.1.3.4 *Body of deregister request*

The SMA gateway **MUST** include the reason for deregistration in the body of the deregister request, as specified in A.4. The Event Server **MUST** include the reason for the deregistration request in the body of the deregister request, as specified in A.4. The Event Server **SHOULD** also include the wait time (i.e., how long to wait for next registration attempt). The Event Server **MAY** include the URIs in the case it is redirecting the SMA gateway. When the optional reconnection information is provided, the SMA gateway **MUST** act on the values provided.

6.1.4 **Heartbeat**

The presence of a TLS or TCP connection between the SMA gateway and the Event Server is a general indication of the presence of a communication channel. However, it does not guarantee that signaling messages will be received by either entity. This is accomplished by a special event message, termed heartbeat.

The heartbeat message is a simple request and response mechanism. When the SMA gateway wants to verify the presence of an Event Server, it sends an event with the URI:

`http://<Event Server FQDN>/ev/HB`

When the Event Server wants to verify the presence of the SMA gateway, it sends an event with the URI:

`http://<SMA gateway FQDN>/ev/HB`

The SMA gateway MUST wait for an operator configured period after the last successful communication it has with the Event Server, prior to sending a heartbeat event. The time of last successful communication is specified as the point in time when the SMA gateway received a request (e.g., HTTP GET) or a response (e.g., 2xx) from the Event Server. For the configuration parameter, refer to [PKT-SMA-PROV]. If a transmitted Heartbeat message does not receive a response, then the SMA gateway MUST retry as it would any other event and indicate the retry attempt using the data elements as specified in Annex A.6. The receipt of a 200 OK response indicates that the Event Server is capable of handling SMA signaling; the SMA gateway MUST treat all other responses as errors. Further, the absence of a 200 OK response, even after retries, MUST be addressed as a connection failure by the SMA gateway.

The Event Server MUST adhere to the same requirements as the SMA gateway, with the roles reversed. However, the configuration of the time-to-wait is out of scope for this document.

6.1.5 Error Handling

This section contains the SMA messaging error conditions and their handling at the Event Server and at the SMA gateway. The Event Server's behavior upon receiving non-2xx HTTP responses from the SMA gateway is specified in Table 1. The SMA gateway's behavior upon receiving non-2xx HTTP responses from the Event Server is specified in Table 2.

Table 1 - Event Server behavior upon receiving HTTP responses from the SMA gateway

HTTP Response from SMA gateway	Required Event Server Action	Suggested action detail
HTTP 3xx responses	Event Server ignores and retries.	Event Server: Retry using backoff algorithm. Log event, send alarm. SMA gateway: Log event
HTTP 400, Bad Request	Event Server ignores and retry.	Event Server: Retry using backoff algorithm. Log event, send alarm SMA gateway: Log event
HTTP 401, Unauthorized	Event Server ignores and retries.	Event Server: Retry using backoff algorithm. Log event, send alarm. SMA gateway: Log event.
HTTP 407, Proxy Authentication Required	Event Server ignores and retries.	Event Server: Log event, if unsuccessful send alarm. SMA gateway: Log event,
HTTP 408, Request Time-out	Event Server retries.	Event Server: Retry using backoff algorithm. Log event. SMA gateway: Log event.
HTTP 488, Not Acceptable Here	Event Server verifies SMA gateway URI	Event Server: Log event, verify SMA gateway URI
HTTP 503, Service Unavailable	Event Server retries.	Event Server: Retry using backoff algorithm. Log event. SMA gateway: Log event.

Table 2 - SMA gateway behavior upon receiving HTTP responses from the Event Server

HTTP Response from Event Server	Required SMA gateway Action	Suggested action detail
HTTP 301, Moved Permanently	SMA gateway retries to new URI	SMA gateway: Store and use new URI, log event.
HTTP 305, Use Proxy	SMA gateway, if authorized, retries to proxy	SMA gateway: Store and use new URI, log event.
HTTP 307, Temporary Redirect	SMA gateway retries with a different Event Server URI.	SMA gateway: Store and use new URI, log event.
HTTP 400, Bad Request	SMA gateway retries.	SMA gateway: Retry one time, log event, continue operation as appropriate (define behavior on repeated error responses). Indicate error condition to user. Event Server: Log event.

HTTP Response from Event Server	Required SMA gateway Action	Suggested action detail
HTTP 401, Unauthorized	SMA gateway performs authentication as specified in Section 6.1.3.1	SMA gateway: Log event, indicate error condition to user. Retry with proper authentication. Event Server: Log event.
HTTP 407, Proxy Authentication Required	SMA gateway performs authentication as specified in Section 6.1.3.1	SMA gateway: Log event, if unsuccessful, indicate error condition to user. Event Server: Log event.
HTTP 408, Request Time-out	SMA gateway retries.	SMA gateway: Retry with backoff algorithm. If persistent, indicate error condition to user. Try on backup channel with appropriate backoff algorithm.
HTTP 488, Not Acceptable Here	SMA gateway verifies Event Server URI	If the Event Server URI matches what is configured, the SMA gateway logs an event. If not, the SMA gateway retries with the correct EventServerURI (which may be different from the SignalingURI used for session establishment).
HTTP 503, Service Unavailable	SMA gateway retries.	SMA gateway: Log event. Retry with backoff algorithm. If persistent, indicate error condition to user. Try on backup channel with appropriate backoff algorithm. Event Server: Log event.

When receiving an HTTP response code listed in Table 1, the Event Server **MUST** take the action listed in the table corresponding to that response code. When receiving an HTTP response code listed in Table 2, the SMA gateway **MUST** take the action listed in the table corresponding to that response code.

6.1.6 Backoff and Retry Procedures

In support of the continued proper operation of the SMA Service and the connected SMA devices, the SMA gateway **MUST** maintain communication with the SMA Event Server. The system could be particularly vulnerable during communication recovery situations including lost messages or network outages. The objectives are to minimize any periods of service unavailability due to network problems, to prevent mass recovery situations from overwhelming the network, and to keep the end user aware of the state of connection.

The following sections indicate the factors that are to be considered and balanced for system operation.

6.1.6.1 Maintaining Security

Security is to be maintained at the user premise. The requirements to support this are:

- The SMA gateway **MUST** use a connection recovery procedure that will not interfere with or compromise any security mechanisms.
- The SMA gateway **SHOULD** log connectivity events including loss of communication with the Event Server retry attempts and reconnections.

6.1.6.2 User Experience

The SMA gateway **MUST** automatically attempt to re-establish communication with the Event Server in the event of error or network disconnection. If available, the SMA gateway **SHOULD** make use of a backup communications channel, if it is determined that the primary communications channel is unavailable. The SMA gateway **SHOULD** indicate to the user, if possible, the state of communication with the SMA Event Server.

6.1.6.3 Behavior after area-wide outage

In order to control the reconnection and re-registration of SMA gateways after an area-wide communication outage:

- The SMA gateway retry mechanism SHOULD include randomized retry intervals.
- The SMA gateway retry mechanism SHOULD have an exponential retry fall-off rate.
- The SMA gateway retry mechanism SHOULD have a maximum retry duration (e.g., 8, 12, 24 hours).

6.1.6.4 Long-term service shutdown

To provide the ability to support long-term service shutdown without dramatically affecting the performance of the underlying network:

- The SMA gateway SHOULD have a limit on the time during which it will try to re-establish communication with the Event Server (e.g., hours or days) after which the SMA gateway will enter a self-disconnected state.
- The SMA gateway SHOULD have a manual forced reconnect function to allow a user to re-establish communication with the Event Server after the self-disconnected state has been entered.

6.1.7 SMA Logic

To allow for efficient controls over the SMA devices in the Home Domain, this document specifies logical constructs that contain conditions and instructions. This is termed as SMA Logic. The specification language introduces the following:

- Macro
- Trigger
- Schedule

SMA Logic requirements are specified in detail in Annex B. The SMA gateway MUST accept valid SMA logical constructs when provided as part of the configuration, or as an update in an instruction. The SMA gateway MAY reject invalid SMA logical constructs.

6.2 Media

PacketCable SMA supports both video and audio streams, also referred to as media sessions. Media sessions can be requested by users who are within the home domain or outside of the home domain. The scope of this specification is limited to the media sessions that are facilitated by signaling between the Event Server and the SMA gateway. While it is possible for SMA devices to directly speak to media clients, it is out of scope for this specification.

Figure 3 shows the network elements that participate in the establishment and maintenance of media sessions. The pkt-sma-1 interface is used to convey SMA signaling messages. The pkt-sma-2 and pkt-sma-3 interfaces are used to convey Simple Traversal of UDP through NAT (STUN) and Traversal Using Relay NAT (TURN) messages used in the traversal of NAT devices.

A media client (e.g., web UI, monitoring station) may request media from a media enabled SMA device via the SMA gateway. This can be either be an ad-hoc request (e.g., user wants a camera to stream) or the result of a trigger (e.g., motion sensor is triggered; user wants to contact emergency personnel). If the media is not being streamed, i.e., a media session is not required, it can be obtained via the SMA signaling interface, e.g., HTTP GET method. If a media session is required (e.g., one-way audio, two-way video), then it is established via a special event request for media and the Session Description Protocol (SDP). Figure 5 illustrates how this is accomplished.

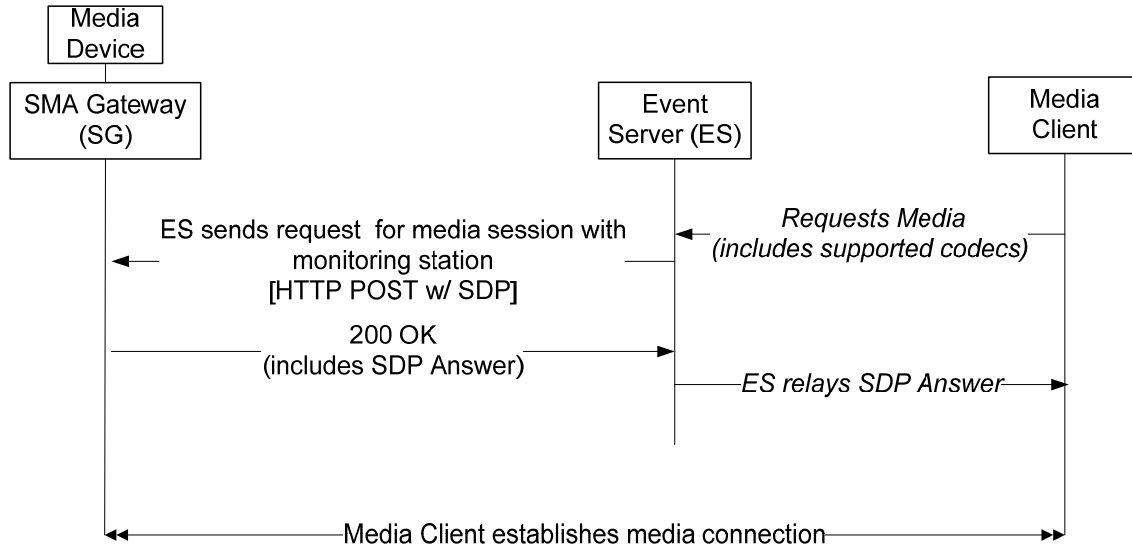


Figure 5 - SMA Media Session Establishment

As indicated in Figure 5, the following steps occur:

- The media client (e.g., web UI or a mobile client) requests media through the Event Server.
- The Event Server sends the request to the SMA gateway via a HTTP POST request, with the required SDP ("offer").
- The SMA gateway requests media from the SMA device.
- Based on the response from the SMA device the SMA gateway creates a response, and includes the SDP ("answer").
- The Event Server relays this back to the media client.
- The media client and the SMA gateway establish a media connection based on the negotiated parameters.

The interface between the Media Client and the Event Server, and the interface between the SMA gateway and the media-enabled SMA device is out of scope for this specification.

For media session establishment, SDP is required. The Event Server **MUST** support the Session Description Protocol (SDP) and the offer/answer model for media session establishment. The SMA gateway **MUST** also support SDP and offer/answer model for multimedia sessions. For more information on SDP, refer to [RFC 4566]. For more information on the Offer/Answer model, see [RFC 3264]. In SMA signaling, the SDP offer is carried inside an HTTP request, and the SDP answer is carried inside the corresponding HTTP response.

The SMA gateway **MUST** be capable for generating SDPs for all media enabled SMA devices. The SMA gateway can either create the SDPs for the various media enabled SMA devices under its control, or if the devices support it, the SMA gateway **MAY** query the device for its capabilities. The SMA gateway **MUST** generate SDP that allows media to traverse a NAT device between the SMA gateway and the access network, if such a NAT device exists.

The Event Server **MUST** pass through any SDP from the originator, modifying the SDP as appropriate to allow media to flow as needed between the two media endpoints. If the media session is initiated by the media-enabled SMA device, the SMA gateway **MUST** pass through any SDP from the originator, modifying the SDP as appropriate to allow media to flow as needed between the two media endpoints.

The SMA gateway MUST proxy media streams from SMA devices in the home domain, when it is involved in media session negotiation. Refer to [PKT-CODEC-MEDIA] for more information on the use of SDP and the Offer/Answer model.

6.2.1 HTTP URI usage

For media session establishment, the Event Server MUST use the HTTP POST method as described in Section 6. The Event Server MUST include an instruction requesting the media session within the HTTP POST request, as indicated below:

`http://<SMA FQDN>/<Device ID>/ev/media/[Start|Pause|Resume|Stop]`, where:

- Start indicates a request to initiate a session (requires an SDP).
- Stop indicates a request to stop an existing session (no SDP is contained in this instruction).
- Pause indicates a request to pause an existing session (no SDP is contained in this instruction).
- Resume indicates a request to resume a session (no SDP is contained in this instruction).

When the Stop instruction is called on a stream, the stream cannot be restarted; however, a new stream can be started using the Start instruction. When the Pause instruction is called on a stream, the stream may be resumed using the Resume instruction; the stream will then use the previously negotiated SDP for the stream. The Pause instruction suppresses streaming until the Resume instruction is called. When the stream is resumed, the stream continues in real-time; no time shifting or PVR functionality is enabled using the Pause and Resume instructions. Some media devices may lack support for Pause and Resume. In such cases, the Pause and Resume instructions will result in failure responses.

The Event Server MUST include a Session Description Protocol within the body of the Start message, describing the media attributes. This SDP will constitute the "offer" as defined in [RFC 3264]. In response, the SMA gateway MUST include an SDP that constitutes the "answer", in the response.

6.2.2 Transport

The SMA gateway MUST support RTP and RTCP as specified in [PKT-CODEC-MEDIA]. SMA gateways MUST also support requirements for supporting QoS and NAT traversal considerations, such as transmitting their RTP stream from the same IP address and port in which it has advertised to receive RTP on in its SDP. The SMA gateway MAY support other transport protocols.

6.2.3 Codecs

The SMA gateway MUST support G.711 and MPEG-4 Part 2, as specified in [PKT-CODEC-MEDIA]. The SMA gateway MAY support other codecs.

6.2.4 NAT and Firewall traversal

PacketCable SMA allows for the SMA gateway to connect via home routers. Home routers can contain NATs, and potentially, firewalls. This can be a hindrance to media session establishment. To address such cases, SMA gateways that can connect via home routers MUST support Internet Connectivity Establishment (ICE) as specified in [ID ICE]. Refer to [TR-PKT-NFT] for more information on NAT traversal using ICE. For an overview of using ICE in non-SIP environments, please refer to [ID-ICE-NONSIP]. Further, PacketCable SMA specifies two interfaces, pkt-sma-2 to the STUN Server ([RFC 5389]), and pkt-sma-3 to the TURN Server ([RFC 5389]), as indicated in Figure 6. The SMA gateway MUST support the two interfaces: pkt-sma-2 and pkt-sma-3.

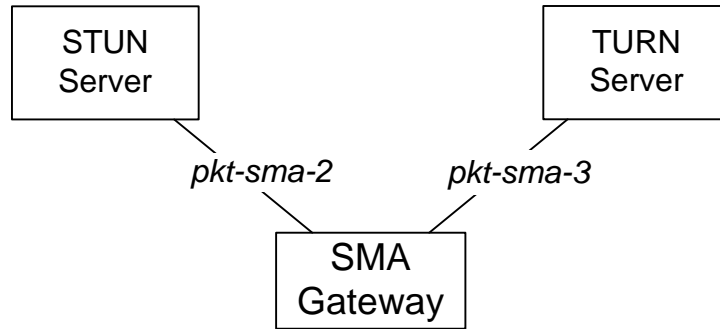


Figure 6 - Interfaces to support media in the presence of NATs

The use of a TURN server allows for the case when the media clients (e.g., web UI) do not support ICE.

6.3 Quality of Service

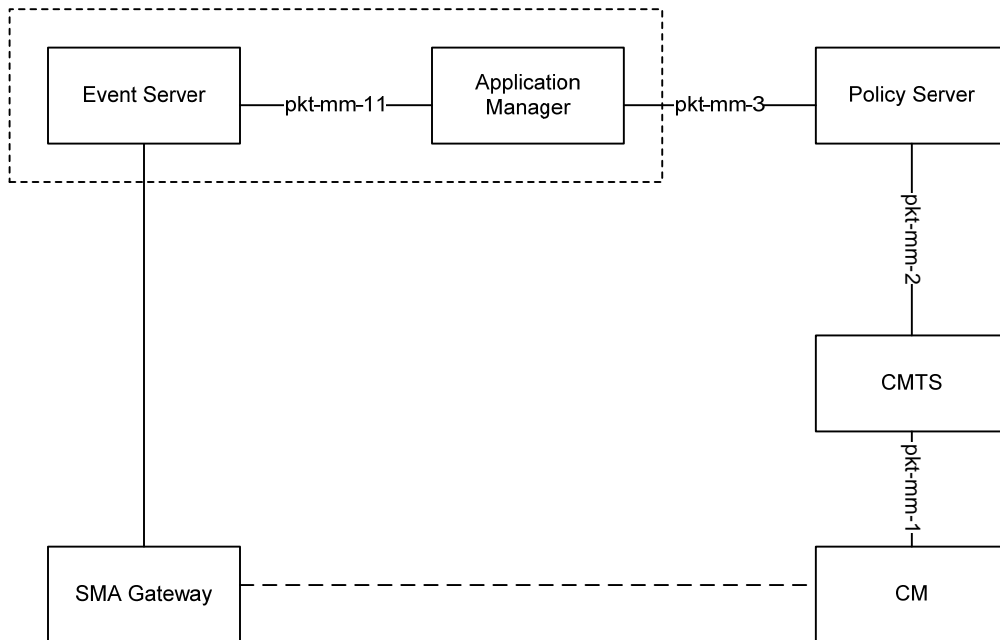


Figure 7 - SMA QoS Interfaces

The QoS mechanism used by SMA is based on the PacketCable Multimedia Specification as defined in [PKT-MM]. Figure 7 shows the functional components of PCMM as they apply to PacketCable SMA. Table 3 describes these interfaces briefly.

Table 3 - QoS Interface Descriptions

Reference Point	PacketCable Network Elements	Reference Point Description
pkt-mm-1	CM – CMTS	The Cable Modem (CM) may request QoS from the CMTS via DOCSIS 1.1 DSx signaling. Alternatively, the CMTS may instruct the CM to setup, teardown or change a DOCSIS service flow in order to satisfy a QoS request, again via DSx signaling.
pkt-mm-2	Policy Server – CMTS	This interface is fundamental to the policy-management framework. It controls policy decisions, which may be: (a) pushed by the Policy Server (PS) onto the CMTS, or (b) pulled from the PS by the CMTS. The interface also allows for proxied QoS requests on behalf of a client. In some scenarios, this interface may also be used to inform the PS when QoS resources have become inactive.
pkt-mm-3	Application Manager – Policy Server	The Application Manager (AM) may request that the PS install a policy decision on the CMTS on behalf of the client. This interface may also be used to inform the AM of changes in the status of QoS resources.
pkt-mm-11	Application Server (Event Server) – Application Manager	The Application Server (AS) uses this interface to send network resource requests on behalf of the client to the AM. This interface may also be used to notify the AS of changes in the status of the network resources.

6.3.1 QoS Usage

In SMA, QoS is used for streaming sessions, such as a video session between an SMA camera at home and a PC on the Internet that has requested the video session. QoS resources for SMA signaling may be dynamically established by the Event Server for each registered SMA gateway, or statically configured in the CM serving the SMA gateway through the CM configuration file. The approach chosen is based on operator decision. Dynamic establishment of QoS resource for SMA signaling follows the same procedures as any other QoS resource request as described in Section 6.3.2 below.

The selection of the type of QoS used for a particular streaming session is determined by the Event Server. The Event Server may use the type of streaming session in its determination of what type of QoS to request. For example, the Event Server may choose Unsolicited Grant Service for an audio streaming session comprised of a fixed-rate audio codec. However, for a video streaming session comprised of a variable-rate audio codec, the Event Server may choose Real-Time Polling Service instead.

Some sessions may change their characteristics during the lifetime of the session. For example, a session that began as an audio session with a fixed-rate codec (such as G.711) may later transition to a variable-rate codec (such as G.729.1). It is the responsibility of the Event Server to monitor such changes and modify the QoS resources appropriately. Likewise, when a session ends, it is the responsibility of the Event Server to relinquish QoS resources upon session termination.

6.3.2 Procedures

The Event Server **MUST** support the pkt-mm-11 interface as defined in [PKT-MM-WS]. The Application Manager **MUST** support the pkt-mm-3 interface as defined in [PKT-MM].

The ContextID used to configure QoS resources by the Application Server must be the same or an extended version of the same (extended via new idExtension elements) for all Web Services messages associated with a particular streaming session, as denoted in section 6.2.1.2 of [PKT-MM-WS]. If multiple streaming sessions exist concurrently for a particular SMA gateway, then each streaming session **MUST** have a unique ContextID.

6.3.2.1 *Session Establishment*

During the initiation of a streaming session, the Event Server **MUST** request QoS resources using either a single-phase or two-phase approach. If the two-phase approach is chosen, the Event Server **MUST** request QoS resources from the Application Manager using the ReserveResources web service message. After successfully reserving QoS resources, the Event Server **SHOULD** request events for the previously requested QoS resources by subscribing to the QoSChangeEvent. Upon the successful initiation of the streaming session, the Event Server **MUST** request QoS resource activation using the CommitResources web service message.

Alternatively, if the single-phase approach is chosen, the Event Server **MUST** use a CommitResources web service message to reserve and commit the QoS resources.

The Event Server **MUST** define an appropriate Classifier to describe the traffic associated with the streaming session and pass that Classifier value in the ReserveResources web service message or the CommitResources web service message as per the procedures defined in [PKT-MM-WS].

6.3.2.2 *Session Modification*

If the parameters of a session change during the lifetime of the session in such a way that QoS resources need to be modified, the Event Server **MUST** send an appropriate ReserveResources web service message or an appropriate CommitResources web service message. For both messages, the EventServer **MUST** ensure that the ServiceName is the same as the ServiceName used in the original session.

6.3.2.3 *Session Termination*

Upon termination of the session, the Event Server **MUST** send an appropriate ReleaseResources web service message to the Application Manager.

6.3.2.4 *Network-Initiated Change of QoS Resources*

If the Event Server receives notification that the network has changed previously reserved QoS resources for an active streaming session, it **MUST** take action appropriate for the type of SMA resource associated with that QoS flow. Specifically, depending on the specific event and type of streaming service, the EventServer may terminate the stream completely using appropriate SMA signaling, or may allow the stream to continue without QoS protection.

6.4 Security

As mentioned in Section 5.3.4, SMA Security is accomplished using TLS. The usage of TLS is determined by configuration information. An SMA gateway that is configured to use TLS for communication **MUST** setup a TLS connection with the Event Server before exchanging any SMA messages. An Event Server **MUST** reject all requests that arrive outside of TLS from an SMA gateway that was configured to use TLS.

For the purposes of this discussion, the SMA gateway is considered to be the TLS Client. The Event Server is considered to be the TLS Server.

6.4.1 Authentication

To prevent the threats discussion in Section 5.3.4, it is important to perform mutual authentication between the SMA gateway and the Event Server. Authentication of the Event Server by the SMA gateway is performed by validating the certificate provided by the server during TLS setup. Authentication of the SMA gateway by the Event Server is performed through the SMA registration procedures according to Section 6.1.3. In order to protect against a man-in-the-middle-attack during client authentication, it is important that the SMA registration be done within a TLS connection.

6.4.2 Message Integrity and Protection

When TLS is used to protect messages exchanged between the SMA gateway and the Event Server, the following integrity mechanisms are provided for TLS based access security:

- Negotiation of TLS related integrity protection and encryption features MUST take place at the TLS layer.
- The SMA gateway MUST always offer TLS cipher suites to be used for the session, as specified in [RFC 4346].
- The Event Server MUST decide which TLS cipher suites are used, as specified in [RFC 4346].
- The SMA gateway MUST verify that the data is sent and received according to [RFC 4346]. This verification is also used to detect if the received data has been tampered with.
- The Event Server MUST verify that the data is sent and received according to [RFC 4346]. This verification is also used to detect if the received data has been tampered with.

6.4.3 TLS Profile

The following requirements apply for the TLS session:

- TLS_RSA_WITH_3DES_EDE_CBC_SHA and the TLS_RSA_WITH_AES_128_CBC_SHA cipher suites will be supported (per [RFC 3268]).
- NULL integrity protection will not be supported.
- NULL encryption may be supported. However, NULL encryption should not be used unless privacy can be ensured by means outside of SMA.
- Anonymous key exchange will not be supported.
- A client certificate may be requested in the Server Hello message.

The SMA gateway MUST adhere to the requirements above pertaining to the TLS session. The Event Server MUST adhere to the requirements above pertaining to the TLS session.

6.4.4 TLS Certificate Profile and Validation

This section specifies the TLS certificate profile, validation, and revocation requirements.

6.4.4.1 TLS Certificate Profiles

The Event Server MUST present X.509 digital certificates [RFC 5280] for authentication in TLS, as profiled in Table 4.

Table 4 - TLS Certificate Profile

TLS Server Certificates	
Subject Name Form	C=<Country> O=<Company> CN=<FQDN> Additional fields may be present in the subject name. FQDN is the Event Server’s fully qualified domain name (e.g., es.example.com). Only a single FQDN is allowed in the CN field.

TLS Server Certificates	
Intended Usage	These certificates are used to authenticate TLS handshake exchanges (and encrypt when using RSA key exchange).
Validity Period	Set by operator policy
Modulus Length	1024, 1536, 2048
Extensions	KeyUsage[critical](digitalSignature, keyEncipherment) extendedKeyUsage (id-kp-serverAuth, id-kp-clientAuth) authorityKeyIdentifier (keyIdentifier=<subjectKeyIdentifier value from CA cert>)

SMA gateway certificates, when available, **MUST** follow the MTA Device certificate profile specified in [PKT-SEC1.5]. SMA gateway certificates **MUST** be issued by the Manufacturer Certificate Authority as specified in [PKT-SEC1.5].

6.4.4.2 Certificate validation

The SMA gateway **MUST** verify that the Event Server's TLS certificates are part of a certificate chain that chains up to a pre-configured CableLabs Root certificate. The chain may contain intermediate Certification Authority (CA) certificates.

Usually the first certificate in the chain is not explicitly included in the certificate chain that is sent by the Event Server to the SMA gateway. In the cases where the first certificate is explicitly included, it is already be known to the verifying party ahead of time and will not contain any changes to the certificate, with the possible exception of certificate serial number, validity period, and the value of the signature. If changes other than the certificate serial number, validity period and the value of the signature exist in the root certificate that was sent by the Event Server to the SMA gateway in comparison to the known root certificate, the SMA gateway **MUST** conclude that the certificate verification has failed.

SMA gateways **MUST** build the certificate chain and validate the TLS certificate according to the "Certificate Path Validation" procedures described in [RFC 5280]. In general, X.509 certificates support a liberal set of rules for determining if the issuer name of a certificate matches the subject name of another. The rules are such that two name fields may be declared to match even though a binary comparison of the two name fields does not indicate a match. [RFC 5280] recommends that certificate authorities restrict the encoding of name fields so that an implementation can declare a match or mismatch using simple binary comparison. Accordingly, the DER-encoded `tbsCertificate.issuer` field of a certificate will be an exact match to the DER-encoded `tbsCertificate.subject` field of its issuer certificate. An implementation may compare an issuer name to a subject name by performing a binary comparison of the DER-encoded `tbsCertificate.issuer` and `tbsCertificate.subject` fields. The SMA gateway **MUST** also validate that the event server FQDN (within the `EventServerURI` or `SignalingURI`) as specified in [RFC 2818], Section 3.1.

6.4.4.3 Certificate Revocation

Certificate Revocation Lists (CRLs) may be checked as part of certificate path validation. The CRL profile and how an SMA gateway obtains a CRL is not specified in this document.

Annex A SMA DATA MODELS (Normative)

This section contains the XML Schemas for the various data models used by this specification. The XML Schemas can also be obtained from the CableLabs website using the following conversion:

- "urn:cablelabs:packetcable:sma:xsd" refers to the HTTP URI <http://www.cablelabs.com/packetcable/sma/xsd/>
- "<version number>" refers to a sub-directory of the same name (e.g., v1)
- "<Schema Identifier>" corresponds to the file "SMA<Schema Identifier>.xsd" (all lower case)

For example, the XML Schema for instruction has the following URN:

- urn:cablelabs:packetcable:sma:xsd:v1:instruction

and the XML Schema can be obtained from:

- <http://www.cablelabs.com/packetcable/sma/xsd/v1/instruction.xsd>

A.1 Instruction

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsd:schema targetNamespace="urn:cablelabs:packetcable:sma:xsd:v1:instruction"
  elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="I">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="##other" processContents="strict"
          minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>

      <!-- This is the correlation ID-->
      <xsd:attribute name="cid" type="xsd:string" use="required" />

      <!-- Timer T1 -->
      <xsd:attribute name="t1" type="xsd:int" use="optional" />

      <!-- Timer T2 -->
      <xsd:attribute name="t2" type="xsd:int" use="optional" />

    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

A.2 Event

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsd:schema targetNamespace="urn:cablelabs:packetcable:sma:xsd:v1:event"
  elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="Ev">
    <xsd:complexType>

      <xsd:sequence>

        <xsd:element name="cat" type="xsd:string" minOccurs="0"/>

      </xsd:sequence>

    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

```

    <!-- This allows us to insert elements from the planned device profiles -->
    <xsd:any namespace="##other" processContents="strict"
             minOccurs="0" maxOccurs="unbounded" />

</xsd:sequence>

    <!-- For extensibility, e.g., vendor-specific extensions-->
    <xsd:anyAttribute namespace="##any" processContents="skip" />
</xsd:complexType>
</xsd:element>

</xsd:schema>

```

A.3 Registration

```

<?xml version="1.0"?>
<xsd:schema
    targetNamespace="urn:cablelabs:packetcable:sma:xsd:v1:reg"
    elementFormDefault="qualified"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:complexType name="reg">
        <xsd:attribute name="gwURI" type="xsd:anyURI" use="required"/>
        <xsd:attribute name="manu" type="xsd:string" use="required"/>
        <xsd:attribute name="model" type="xsd:string" use="required"/>
        <xsd:attribute name="version" type="xsd:string" use="required"/>

        <!-- Comma separated list of versions. For example:
             - "1.0" indicates only one version (1.0).
             - "1.0,1.1" indicates support for 1.0 and 1.1
             The current version is 1.0 -->
        <xsd:attribute name="supportedProtocols" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:schema>

```

A.4 Deregistration

```

<?xml version="1.0"?>

<xsd:schema targetNamespace="urn:cablelabs:packetcable:sma:xsd:v1:dereg"
    elementFormDefault="qualified"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:complexType name="dereg">
        <xsd:attribute name="reason" type="xsd:string" use="required"/>
        <xsd:attribute name="delay" type="xsd:integer" use="optional"/>
        <!-- The following URIs are only used when an Event Server is asking the
             SMA gateway to register with a different Event Server.
             This will replace any configured values of EventServerURI and
             SignalingURI. The same rules are the configured values are used
             for connection establishment.
             -->
        <xsd:attribute name="newEventServerURI" type="xsd:anyURI" use="optional"/>
        <xsd:attribute name="newSignalingURI" type="xsd:anyURI" use="optional"/>

        <!-- This is the time prior to establishing a new connection with the
             same or different event server. If the above URIs are included
             it can contain a new event server. If not, the SMA gateway will
             wait for the following duration prior to attempting a TLS or TCP
             connection with the event server directing deregistration. -->
        <xsd:attribute name="waitTime" type="xsd:unsignedInt" use="optional"/>
    </xsd:complexType>
</xsd:schema>

```

A.5 Response

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsd:schema targetNamespace="urn:cablelabs:packetcable:sma:xsd:v1:devprofile:response"
  elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:this="urn:cablelabs:packetcable:sma:xsd:v1:devprofile:response">

  <xsd:element name="R">
    <xsd:complexType>
      <xsd:sequence>
        <!-- Property:Value pairs can be returned using this construct-->
        <xsd:element minOccurs="0" maxOccurs="unbounded" ref="this:p"/>
        <xsd:any namespace="##other" processContents="strict"
          minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <!-- This is the correlation ID-->
      <xsd:attribute name="cid" type="xsd:string" use="optional" />

      <!-- Instruction processing result: true(success), false(failure)-->
      <xsd:attribute name="res" type="xsd:boolean" use="required" />
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="p">
    <xsd:complexType>
      <xsd:attribute name="property" use="required" type="xsd:string"/>
      <xsd:attribute name="value" use="required" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

A.6 Device List

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sd="urn:cablelabs:packetcable:sma:xsd:v1:deviceprofile"
  xmlns:dl="urn:cablelabs:packetcable:sma:xsd:v1:devlist"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  targetNamespace="urn:cablelabs:packetcable:sma:xsd:v1:devlist">

  <xs:import namespace="urn:cablelabs:packetcable:sma:xsd:v1:deviceprofile"
    schemaLocation="http://www.cablelabs.com/packetcable/sma/xsd/v1/deviceprofile.xsd"/>

  <xs:element name="deviceList">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="dl:dev"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="dev">
    <xs:complexType>
      <xs:attribute name="id" use="required" type="xs:string"/>
      <xs:attribute name="name" use="required" type="xs:string"/>
      <xs:attribute name="tag" use="optional" type="sd:TagType"/>
      <xs:attribute name="ns" use="required" type="xs:anyURI"/>
      <xs:attribute name="nsloc" use="optional" type="xs:anyURI"/>
    </xs:complexType>
  </xs:element>
```

```

    </xs:complexType>
  </xs:element>
</xs:schema>

```

A.7 Heartbeat

```

<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema targetNamespace="urn:cablelabs:packetcable:sma:xsd:v1:heartbeat"
  elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="hb">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="##other" processContents="strict" minOccurs="0"
          maxOccurs="unbounded" />
      </xsd:sequence>

      <xsd:attribute name="r" type="xsd:integer" use="optional">
        <xsd:annotation>
          <xsd:documentation>
            This attribute is only used for retry attempts within a Heartbeat
            cycle. For the first heartbeat message, this attribute is not
            required. If the first heartbeat message is not acknowledged with
            a valid response, then this attribute is included in subsequent
            retry attempts. Further, this attribute will be set to a value
            of '1' for the first retry, and incremented by '1' in subsequent
            retry attempts within the same Heartbeat cycle.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
      <xsd:attribute name="m" type="xsd:integer" use="optional">
        <xsd:annotation>
          <xsd:documentation>
            This attribute is only used when the transmitting entity is
            reaching the maximum number of retries before which it will
            declare a connection failure. It is included in the heartbeat
            message prior to the last retry, and the last retry attempt.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
      <xsd:attribute name="t1" type="xsd:integer" use="optional">
        <xsd:annotation>
          <xsd:documentation>
            This attribute indicates the time within which the transmitting
            entity is expecting a response. If the receiving entity does not
            respond with a HTTP response within t1, it will be considered
            a failure. When the SMA Gateway is the receiving entity,
            if this attribute is omitted, it will use the configured timer 't1'.
          </xsd:documentation>
        </xsd:annotation>
      </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>

```

A.8 Logic

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:SL="urn:cablelabs:packetcable:sma:xsd:v1:smalogic"
  targetNamespace="urn:cablelabs:packetcable:sma:xsd:v1:smalogic">
  <!-- TYPE DECLARATIONS -->
  <xs:complexType name="ExpressionType" abstract="true"/>
  <xs:complexType name="MacroType">

```

```

<xs:sequence>
  <xs:element ref="SL:statement" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="TriggerType">
  <xs:sequence>
    <xs:element ref="SL:booleanExpression"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
</xs:complexType>
<xs:simpleType name="DowType">
  <xs:restriction base="xs:string">
    <xs:pattern value="(s|S)*(m|M)*(t|T)*(w|W)*(r|R)*(f|F)*(a|A)*"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TodType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-2][0-9][0-9][0-9][0-9][0-9]"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="AbsoluteTimeType">
  <xs:annotation>
    <xs:documentation>
      Specifies an absolute time as the number of seconds since
      January 1, 1970 UTC.
    </xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>
<xs:complexType name="ScheduleType">
  <xs:sequence>
    <xs:element name="DayEq" type="SL:DowBooleanOperatorType" minOccurs="0"/>
    <xs:element name="TimeEq" type="SL:TodBooleanOperatorType" minOccurs="0"/>
    <xs:element name="Before" type="SL:TodBooleanOperatorType" minOccurs="0"/>
    <xs:element name="After" type="SL:TodBooleanOperatorType" minOccurs="0"/>
    <xs:element name="Every" type="xs:nonNegativeInteger" minOccurs="0"/>
    <xs:element name="At" type="SL:AbsoluteTimeType" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="OperatorType">
  <xs:complexContent>
    <xs:extension base="SL:ExpressionType">
      <xs:sequence>
        <xs:element ref="SL:expression" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="type" type="SL:AssignmentTargetType" use="required"/>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="value" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="SetOperatorType">
  <xs:complexContent>
    <xs:extension base="SL:ExpressionType">
      <xs:sequence>
        <xs:element ref="SL:expression" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="type" type="SL:AssignmentTargetType" use="required"/>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="value" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

```

<xs:complexType name="GetOperatorType">
  <xs:complexContent>
    <xs:extension base="SL:ExpressionType">
      <xs:attribute name="type" type="SL:AssignmentTargetType" use="required"/>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="CallMethodType">
  <xs:complexContent>
    <xs:extension base="SL:ExpressionType">
      <xs:sequence>
        <xs:element ref="SL:Param" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="target" type="xs:string" use="required"/>
      <xs:attribute name="method" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="CallMacroType">
  <xs:complexContent>
    <xs:extension base="SL:ExpressionType">
      <xs:sequence>
        <xs:element ref="SL:Param" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="BooleanOperatorType">
  <xs:complexContent>
    <xs:extension base="SL:BooleanExpressionType">
      <xs:choice minOccurs="0">
        <xs:element ref="SL:expression" minOccurs="0"/>
        <xs:element ref="SL:booleanExpression" minOccurs="0"/>
      </xs:choice>
      <xs:attribute name="type" type="SL:AssignmentTargetType" use="required"/>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="value" use="optional"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="TodBooleanOperatorType">
  <xs:complexContent>
    <xs:extension base="SL:BooleanExpressionType">
      <xs:attribute name="tod" type="SL:TodType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="DowBooleanOperatorType">
  <xs:complexContent>
    <xs:extension base="SL:BooleanExpressionType">
      <xs:attribute name="dow" type="SL:DowType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="WaitType">
  <xs:attribute name="period" type="xs:nonNegativeInteger" use="optional"/>
  <xs:attribute name="min" type="xs:nonNegativeInteger" use="optional"/>
  <xs:attribute name="max" type="xs:nonNegativeInteger" use="optional"/>
</xs:complexType>
<xs:complexType name="LogicOperatorType">
  <xs:complexContent>
    <xs:extension base="SL:BooleanExpressionType">
      <xs:sequence minOccurs="2" maxOccurs="unbounded">
        <xs:choice>
          <xs:element ref="SL:booleanExpression"/>
        </xs:choice>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

```

        <xs:element ref="SL:logicOperator" />
      </xs:choice>
    </xs:sequence>
  </xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:simpleType name="aaa">
  <xs:restriction base="xs:string">
    <xs:enumeration value="a" />
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="AssignmentTargetType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="local" />
    <xs:enumeration value="global" />
    <xs:enumeration value="property" />
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="BooleanExpressionType">
  <xs:complexContent>
    <xs:extension base="SL:ExpressionType" />
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="DoerType">
  <xs:sequence>
    <xs:element ref="SL:statement" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="IfType">
  <xs:sequence>
    <xs:element ref="SL:booleanExpression" />
    <xs:element name="Then" />
    <xs:element name="Else" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="WhileUntilType">
  <xs:sequence>
    <xs:element ref="SL:booleanExpression" />
    <xs:element name="Do" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="SwitchType">
  <xs:sequence>
    <xs:element name="Case" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="CaseType">
  <xs:sequence>
    <xs:element ref="SL:booleanExpression" />
    <xs:element ref="SL:statement" minOccurs="0" maxOccurs="unbounded" />
    <xs:element name="Break" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="ParamType">
  <xs:sequence>
    <xs:element ref="SL:expression" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" />
</xs:complexType>
<!-- ELEMENTS -->
<!-- Abstract Elements -->
<xs:element name="statement" abstract="true" />
<xs:element name="expression" type="SL:ExpressionType" abstract="true" />
<xs:element name="booleanExpression" type="SL:BooleanExpressionType"
abstract="true" />
  <xs:element name="logicOperator" type="SL:LogicOperatorType" abstract="true" />
<!-- Conditionals Elements -->

```

```

<xs:element name="If" type="SL:IfType" substitutionGroup="SL:statement"/>
<xs:element name="While" type="SL:WhileUntilType" substitutionGroup="SL:statement"/>
<xs:element name="Until" type="SL:WhileUntilType" substitutionGroup="SL:statement"/>
<xs:element name="Switch" type="SL:SwitchType" substitutionGroup="SL:statement"/>
<!-- PREDEFINED ELEMENTS -->
<xs:element name="DayEq" type="SL:DowBooleanOperatorType"
substitutionGroup="SL:booleanExpression"/>
<xs:element name="TimeEq" type="SL:TodBooleanOperatorType"
substitutionGroup="SL:booleanExpression"/>
<xs:element name="Before" type="SL:TodBooleanOperatorType"
substitutionGroup="SL:booleanExpression"/>
<xs:element name="After" type="SL:TodBooleanOperatorType"
substitutionGroup="SL:booleanExpression"/>
<xs:element name="Equals" type="SL:BooleanOperatorType"
substitutionGroup="SL:booleanExpression"/>
<xs:element name="Less" type="SL:BooleanOperatorType"
substitutionGroup="SL:booleanExpression"/>
<xs:element name="And" type="SL:LogicOperatorType"
substitutionGroup="SL:booleanExpression"/>
<xs:element name="Or" type="SL:LogicOperatorType"
substitutionGroup="SL:booleanExpression"/>
<xs:element name="Not" type="SL:LogicOperatorType"
substitutionGroup="SL:booleanExpression"/>
<xs:element name="Add" type="SL:OperatorType" substitutionGroup="SL:expression"/>
<xs:element name="Subtract" type="SL:OperatorType"
substitutionGroup="SL:expression"/>
<xs:element name="Multiply" type="SL:OperatorType"
substitutionGroup="SL:expression"/>
<xs:element name="Divide" type="SL:OperatorType" substitutionGroup="SL:expression"/>
<xs:element name="Concat" type="SL:OperatorType" substitutionGroup="SL:expression"/>
<xs:element name="Get" type="SL:GetOperatorType" substitutionGroup="SL:expression"/>
<xs:element name="Set" type="SL:SetOperatorType" substitutionGroup="SL:statement"/>
<xs:element name="CallMacro" type="SL:CallMacroType"
substitutionGroup="SL:expression"/>
<xs:element name="CallExternal" type="SL:CallMethodType"
substitutionGroup="SL:expression"/>
<xs:element name="RunMacro" type="SL:CallMacroType"
substitutionGroup="SL:statement"/>
<xs:element name="CallMethod" type="SL:CallMethodType"
substitutionGroup="SL:expression"/>
<xs:element name="RunMethod" type="SL:CallMethodType"
substitutionGroup="SL:statement"/>
<xs:element name="Return" type="SL:GetOperatorType"
substitutionGroup="SL:statement"/>
<xs:element name="Wait" type="SL:WaitType" substitutionGroup="SL:statement"/>
<xs:element name="Then" type="SL:DoerType"/>
<xs:element name="Else" type="SL:DoerType"/>
<xs:element name="Do" type="SL:DoerType"/>
<xs:element name="Case" type="SL:CaseType"/>
<xs:element name="Comment" type="xs:string"/>
<xs:element name="Param" type="SL:ParamType"/>
<!-- Root Elements -->
<xs:element name="Macro" type="SL:MacroType"/>
<xs:element name="Trigger" type="SL:TriggerType"/>
<xs:element name="Schedule" type="SL:ScheduleType"/>
</xs:schema>

```

A.9 Device Profile

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sd="urn:cablelabs:packetcable:sma:xsd:v1:deviceprofile"
  targetNamespace="urn:cablelabs:packetcable:sma:xsd:v1:deviceprofile">
  <xs:simpleType name="TagType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Lighting"/>
    </xs:restriction>
  </xs:simpleType>

```

```

<xs:enumeration value="Climate"/>
<xs:enumeration value="Security"/>
<xs:enumeration value="Surveillance"/>
<xs:enumeration value="Controller"/>
<xs:enumeration value="MediaServer"/>
<xs:enumeration value="HVAC"/>
<xs:enumeration value="Safety"/>
<xs:enumeration value="Light"/>
<xs:enumeration value="Thermostat"/>
<xs:enumeration value="Window Covering"/>
<xs:enumeration value="Fan"/>
<xs:enumeration value="Sensor"/>
<xs:enumeration value="Actuator"/>
<xs:enumeration value="Motion Sensor"/>
<xs:enumeration value="Occupancy Sensor"/>
<xs:enumeration value="Temperature Sensor"/>
<xs:enumeration value="Glass Break sensor"/>
<xs:enumeration value="Door/Window Sensor"/>
<xs:enumeration value="CO Sensor"/>
<xs:enumeration value="Heat Sensor"/>
<xs:enumeration value="Smoke Sensor"/>
<xs:enumeration value="Flammable Gas sensor"/>
<xs:enumeration value="Water Sensor"/>
<xs:enumeration value="Water Level sensor"/>
<xs:enumeration value="Humidity Sensor"/>
<xs:enumeration value="Wind Speed Sensor"/>
<xs:enumeration value="Wind Direction Sensor"/>
<xs:enumeration value="Rain Sensor"/>
<xs:enumeration value="Dewpoint Sensor"/>
<xs:enumeration value="Barometric Pressure Sensor"/>
<xs:enumeration value="Stress Sensor"/>
<xs:enumeration value="Door Lock"/>
<xs:enumeration value="Alarm Panel"/>
<xs:enumeration value="Keypad"/>
<xs:enumeration value="Keyfob"/>
<xs:enumeration value="Panic Pendant"/>
<xs:enumeration value="Camera"/>
<xs:enumeration value="Weight Sensor"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="BaseDataTypes">
  <xs:restriction base="xs:string">
    <xs:enumeration value="string"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="number"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="float"/>
    <xs:enumeration value="second"/>
    <xs:enumeration value="meters">
      <xs:annotation>
        <xs:documentation>
          Length.
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="watts">
      <xs:annotation>
        <xs:documentation>
          Power.
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="psi">
      <xs:annotation>
        <xs:documentation>
          Pressure.
        </xs:documentation>
      </xs:annotation>
    </xs:enumeration>
  </xs:restriction>
</xs:simpleType>

```

```
        </xs:documentation>
    </xs:annotation>
</xs:enumeration>
<xs:enumeration value="percent">
    <xs:annotation>
        <xs:documentation>
            integer, 0-100.
        </xs:documentation>
    </xs:annotation>
</xs:enumeration>
<xs:enumeration value="angularDegrees"/>
<xs:enumeration value="degreesCPerMinute"/>
</xs:restriction>
</xs:simpleType>
<!-- Some enumeration which could be overridden -->
<xs:simpleType name="FanModes">
    <xs:restriction base="xs:string">
        <xs:enumeration value="On"/>
        <xs:enumeration value="Off"/>
        <xs:enumeration value="Low"/>
        <xs:enumeration value="Med"/>
        <xs:enumeration value="Hi"/>
        <xs:enumeration value="Auto"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TemperatureFanModes">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Heat"/>
        <xs:enumeration value="Cool"/>
        <xs:enumeration value="Emergency Heart"/>
        <xs:enumeration value="Off"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ThermostatModes">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Off"/>
        <xs:enumeration value="Heat"/>
        <xs:enumeration value="Cool"/>
        <xs:enumeration value="Auto"/>
        <xs:enumeration value="EmergencyHeat"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SecuritySystemModes">
    <xs:restriction base="xs:string">
        <xs:enumeration value="Armed"/>
        <xs:enumeration value="Disarmed"/>
        <xs:enumeration value="StayArmed"/>
        <xs:enumeration value="Alarming"/>
        <xs:enumeration value="Disconnected"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="BatteryStates">
    <xs:restriction base="xs:string">
        <xs:enumeration value="charging"/>
        <xs:enumeration value="depleted"/>
        <xs:enumeration value="fullycharged"/>
        <xs:enumeration value="unknown"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="CommunicationChannelStatus">
    <xs:restriction base="xs:string">
        <xs:enumeration value="connected"/>
        <xs:enumeration value="disconnected"/>
        <xs:enumeration value="generalError"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TemperatureUnits">
```

```

    <xs:restriction base="xs:string">
      <xs:enumeration value="celsius"/>
      <xs:enumeration value="fahrenheit"/>
    </xs:restriction>
  </xs:simpleType>
</xs:simpleType name="StatusUnits">
  <xs:restriction base="xs:string">
    <xs:enumeration value="On"/>
    <xs:enumeration value="Off"/>
    <xs:enumeration value="Unknown"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="OpeningStates">
  <xs:restriction base="xs:string">
    <xs:enumeration value="open"/>
    <xs:enumeration value="closed"/>
    <xs:enumeration value="partial"/>
    <xs:enumeration value="unknown"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ButtonStates">
  <xs:restriction base="xs:string">
    <xs:enumeration value="pressed"/>
    <xs:enumeration value="released"/>
    <xs:enumeration value="held"/>
    <xs:enumeration value="doubleTapped"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="SpeedUnits">
  <xs:restriction base="xs:string">
    <xs:enumeration value="mph"/>
    <xs:enumeration value="kmph"/>
    <xs:enumeration value="fps"/>
    <xs:enumeration value="mps"/>
    <xs:enumeration value="knots"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="OpenCloseModes">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Closed"/>
    <xs:enumeration value="Opened"/>
    <xs:enumeration value="Partial"/>
    <xs:enumeration value="Unknown"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="enumUnits">
  <xs:union memberTypes="sd:BaseDataTypes sd:FanModes sd:TemperatureFanModes
sd:FanModes sd:TemperatureFanModes sd:ThermostatModes sd:SecuritySystemModes
sd:SecuritySystemModes sd:BatteryStates sd:CommunicationChannelStatus
sd:TemperatureUnits sd:StatusUnits sd:OpeningStates sd:ButtonStates sd:SpeedUnits
sd:OpenCloseModes"/>
</xs:simpleType>
<!-- Properties -->
<xs:complexType name="Property">
  <xs:attribute name="read" type="xs:boolean" use="required"/>
  <xs:attribute name="write" type="xs:boolean" use="required"/>
  <xs:attribute name="unit" type="sd:enumUnits" use="optional"/>
  <xs:attribute name="min" type="xs:double" use="optional"/>
  <xs:attribute name="max" type="xs:double" use="optional"/>
</xs:complexType>
<!-- Signaling -->
<xs:complexType name="PropertyValuePair">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>
<!-- Sensing Properties -->
<xs:complexType name="BinaryInputSensorProperty">

```

```

<xs:complexContent>
  <xs:restriction base="sd:Property">
    <xs:attribute name="read" type="xs:boolean" use="required" fixed="true"/>
    <xs:attribute name="write" type="xs:boolean" use="required" fixed="false"/>
    <xs:attribute name="unit" type="sd:enumUnits" fixed="boolean"/>
    <xs:attribute name="min" use="prohibited"/>
    <xs:attribute name="max" use="prohibited"/>
  </xs:restriction>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="AnalogInputSensorProperty">
  <xs:complexContent>
    <xs:restriction base="sd:Property">
      <xs:attribute name="read" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="write" type="xs:boolean" use="required" fixed="false"/>
      <xs:attribute name="unit" type="sd:enumUnits" fixed="double"/>
      <xs:attribute name="min" type="xs:double" use="required"/>
      <xs:attribute name="max" type="xs:double" use="required"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="TemperatureSensorProperty">
  <xs:complexContent>
    <xs:restriction base="sd:Property">
      <xs:attribute name="read" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="write" type="xs:boolean" use="required" fixed="false"/>
      <xs:attribute name="unit" type="sd:TemperatureUnits" default="fahrenheit"/>
      <xs:attribute name="min" type="xs:double" use="required"/>
      <xs:attribute name="max" type="xs:double" use="required"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<!-- Status Properties -->
<xs:complexType name="Stat">
  <xs:annotation>
    <xs:documentation>
      Three potential values:
      On, Off, Unknown
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:restriction base="sd:Property">
      <xs:attribute name="read" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="write" type="xs:boolean" use="required" fixed="false"/>
      <xs:attribute name="unit" type="sd:StatusUnits" default="Unknown"/>
      <xs:attribute name="min" type="xs:double" use="prohibited"/>
      <xs:attribute name="max" type="xs:double" use="prohibited"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="OnlevelProperty">
  <xs:annotation>
    <xs:documentation>
      Onlevel also includes On/off this way, we do not have to
      have multiple instantiations of the same class
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:restriction base="sd:Property">
      <xs:attribute name="read" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="write" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="unit" type="sd:BaseDataTypes" fixed="percent"/>
      <xs:attribute name="min" type="xs:double" use="optional" default="0"/>
      <xs:attribute name="max" type="xs:double" use="optional" default="100"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

```

```

<xs:complexType name="RampRateProperty">
  <xs:complexContent>
    <xs:restriction base="sd:Property">
      <xs:attribute name="read" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="unit" type="sd:BaseDataTypes" fixed="second"/>
      <xs:attribute name="min" type="xs:double" use="optional" default="0"/>
      <xs:attribute name="max" type="xs:double" use="optional" default="60"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="FanModeProperty">
  <xs:complexContent>
    <xs:restriction base="sd:Property">
      <xs:attribute name="read" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="write" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="unit" type="sd:FanModes" default="Auto"/>
      <xs:attribute name="min" type="xs:double" use="prohibited"/>
      <xs:attribute name="max" type="xs:double" use="prohibited"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="TemperatureModeProperty">
  <xs:complexContent>
    <xs:restriction base="sd:Property">
      <xs:attribute name="read" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="write" type="xs:boolean" use="required" fixed="true"/>
      <xs:attribute name="unit" type="sd:TemperatureFanModes" default="Off"/>
      <xs:attribute name="min" type="xs:double" use="prohibited"/>
      <xs:attribute name="max" type="xs:double" use="prohibited"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ThermostatModeProperty">
  <xs:complexContent>
    <xs:restriction base="sd:Property">
      <xs:attribute name="read" type="xs:boolean" use="required" fixed="false"/>
      <xs:attribute name="unit" type="sd:ThermostatModes" default="Auto"/>
      <xs:attribute name="min" type="xs:double" use="prohibited"/>
      <xs:attribute name="max" type="xs:double" use="prohibited"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="SetpointProperty">
  <xs:complexContent>
    <xs:restriction base="sd:Property">
      <xs:attribute name="read" type="xs:boolean" use="required" fixed="false"/>
      <xs:attribute name="unit" type="sd:TemperatureUnits" default="fahrenheit"/>
      <xs:attribute name="min" type="xs:double" use="optional" default="0"/>
      <xs:attribute name="max" type="xs:double" use="optional" default="114"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
<!-- Methods -->
<!-- Methods return HTTP responses (provisional/standard) -->
<xs:complexType name="Method"/>
<xs:complexType name="DimmerPresetMethod">
  <xs:complexContent>
    <xs:extension base="sd:Method">
      <xs:sequence>
        <xs:element name="brightness" type="sd:BaseDataTypes"
          fixed="percent" minOccurs="0"/>
        <xs:element name="ramprate" type="sd:BaseDataTypes"
          fixed="percent" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

```

<!-- Interfaces -->
<xs:complexType name="Interface" abstract="true">
  <xs:annotation>
    <xs:documentation>
      Defines abstract device functionalities
      All other device profiles shall inherit from Interface
      *note: functionalities and not properties
    </xs:documentation>
  </xs:annotation>
  <xs:attribute name="name" type="xs:string" use="optional"/>
</xs:complexType>
<xs:complexType name="PowerInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="turnOn" type="sd:Method"/>
        <xs:element name="turnOff" type="sd:Method"/>
        <xs:element name="status" type="sd:Stat" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="BinaryInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="state" type="sd:BinaryInputSensorProperty"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="AnalogInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="state" type="sd:AnalogInputSensorProperty"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="AnalogPercentInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="state" type="sd:BaseDataTypes" fixed="percent"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="DimmableInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="turnOnTo" type="sd:DimmerPresetMethod"/>
        <xs:element name="turnOffTo" type="sd:DimmerPresetMethod"/>
        <xs:element name="dim" type="sd:DimmerPresetMethod" minOccurs="0"/>
        <xs:element name="brighten" type="sd:DimmerPresetMethod" minOccurs="0"/>
        <xs:element name="brightnessLevel" type="sd:OnlevelProperty" minOccurs="0"/>
        <xs:element name="brightnessPreset" type="sd:OnlevelProperty"
minOccurs="0"/>
        <xs:element name="rampRatePreset" type="sd:RampRateProperty" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="TempCtrlInterface">
  <xs:complexContent>

```

```

<xs:extension base="sd:Interface">
  <xs:sequence>
    <xs:element name="currentSetpoint" type="sd:SetpointProperty"/>
    <xs:element name="heatSetpoint" type="sd:SetpointProperty" minOccurs="0"/>
    <xs:element name="coolSetpoint" type="sd:SetpointProperty" minOccurs="0"/>
    <xs:element name="incrementSetpoint" type="sd:Method" minOccurs="0"/>
    <xs:element name="decrementSetpoint" type="sd:Method" minOccurs="0"/>
  </xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="ClimateControlInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="fanMode" type="sd:FanModeProperty"/>
        <xs:element name="fanStatus" type="sd:Stat" minOccurs="0"/>
        <xs:element name="temperatureMode" type="sd:FanModeProperty" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="TemperatureInputInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="temperature" type="sd:TemperatureSensorProperty"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="ThermostatInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="temperature" type="sd:TemperatureSensorProperty"/>
        <xs:element name="currentSetpoint" type="sd:SetpointProperty"/>
        <xs:element name="coolSetpoint" type="sd:SetpointProperty" minOccurs="0"/>
        <xs:element name="heatSetpoint" type="sd:SetpointProperty" minOccurs="0"/>
        <xs:element name="fanMode" type="sd:FanModeProperty" minOccurs="0"/>
        <xs:element name="thermostatMode" type="sd:ThermostatModeProperty"
          minOccurs="0"/>
        <xs:element name="incrementSetpoint" type="sd:Method"/>
        <xs:element name="decrementSetpoint" type="sd:Method"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="PanInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="panRight" type="sd:Method"/>
        <xs:element name="panLeft" type="sd:Method"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="TiltInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="tiltUp" type="sd:Method"/>
        <xs:element name="tiltDown" type="sd:Method"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>

```

```

</xs:complexType>
<xs:complexType name="ZoomInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="zoomIn" type="sd:Method"/>
        <xs:element name="zoomOut" type="sd:Method"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="CameraPresetInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="name" type="sd:BaseDataTypes" fixed="string"/>
        <xs:element name="index" type="sd:BaseDataTypes" fixed="integer"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="MediaInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="getMedia" type="sd:Method"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="CoveringInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="open" type="sd:Method"/>
        <xs:element name="close" type="sd:Method"/>
        <xs:element name="level" type="sd:BaseDataTypes"
          fixed="percent" minOccurs="0">
          <xs:annotation>
            <xs:documentation>
              This indicates the percent open, i.e.,
              0 = "Fully Closed"
              100 = "Fully Open"
            </xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="angleOfRotationInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="angle" type="sd:BaseDataTypes" fixed="angularDegrees"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="batteryInterface">
  <xs:complexContent>
    <xs:extension base="sd:Interface">
      <xs:sequence>
        <xs:element name="batteryState" type="sd:BatteryStates"/>
        <xs:element name="powerLevel" type="sd:BaseDataTypes"
          fixed="percent" minOccurs="0"/>
        <xs:element name="powerVoltage" type="sd:BaseDataTypes"

```

```

        fixed="float" minOccurs="0"/>
    </xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="DeviceOnBatteryInterface">
    <xs:complexContent>
        <xs:extension base="sd:Interface">
            <xs:sequence>
                <xs:element name="runningOnBattery" type="sd:BaseDataTypes"
fixed="boolean"/>
                <!-- Is this required? -->
                <xs:element name="runningOnPrimary" type="sd:BaseDataTypes"
fixed="boolean"/>
                <xs:element name="runOnBattery" type="sd:Method" minOccurs="0"/>
                <xs:element name="runOnPrimary" type="sd:Method" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="RainSensorInterface">
    <xs:complexContent>
        <xs:extension base="sd:Interface">
            <xs:sequence>
                <xs:element name="state" type="sd:AnalogInterface"/>
                <xs:element name="reset" type="sd:Method" minOccurs="0"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="DirectionInterface">
    <xs:complexContent>
        <xs:extension base="sd:Interface">
            <xs:choice>
                <xs:element name="azimuthAngle" type="sd:BaseDataTypes"
fixed="angularDegrees"/>
                <xs:element name="azimuthString" type="sd:BaseDataTypes" fixed="string"/>
            </xs:choice>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="SpeedInterface">
    <xs:complexContent>
        <xs:extension base="sd:Interface">
            <xs:sequence>
                <xs:element name="value" type="sd:BaseDataTypes" fixed="float"/>
                <xs:element name="units" type="sd:SpeedUnits" default="mph"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="PressureInterface">
    <xs:complexContent>
        <xs:extension base="sd:Interface">
            <xs:sequence>
                <xs:element name="value" type="sd:BaseDataTypes" fixed="float"/>
                <!-- Do we need an enumeration? -->
                <xs:element name="units" type="sd:BaseDataTypes" fixed="string"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="ButtonInterface">
    <xs:complexContent>
        <xs:extension base="sd:Interface">
            <xs:sequence>
                <xs:element name="id" type="sd:BaseDataTypes" fixed="string"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

```

```

        <xs:element name="state" type="sd:ButtonStates" />
        <xs:element name="press" type="sd:Method" minOccurs="0" />
        <xs:element name="release" type="sd:Method" minOccurs="0" />
    </xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="AlarmPanelInterface">
    <xs:complexContent>
        <xs:extension base="sd:Interface">
            <xs:sequence>
                <xs:element name="state" type="sd:SecuritySystemModes" />
                <xs:element name="ready" type="sd:BaseDataTypes" fixed="boolean" />
                <xs:element name="lastStateChange" type="sd:BaseDataTypes" fixed="string"
                    minOccurs="0" />
                <xs:element name="lastZoneTriggered" type="sd:BaseDataTypes" fixed="string"
                    minOccurs="0" />
                <xs:element name="lastMethod" type="sd:BaseDataTypes" fixed="string"
                    minOccurs="0" />
                <xs:element name="lastMethodStatus" type="sd:BaseDataTypes"
                    fixed="string" minOccurs="0" />
                <xs:element name="arm" type="sd:Method" minOccurs="0" />
                <xs:element name="disarm" type="sd:Method" minOccurs="0" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="CommunicationChannelInterface">
    <xs:complexContent>
        <xs:extension base="sd:Interface">
            <xs:sequence>
                <xs:element name="state" type="sd:CommunicationChannelStatus" />
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<xs:complexType name="Device">
    <xs:sequence>
        <xs:element name="interface" type="sd:Interface" minOccurs="0"
            maxOccurs="unbounded" />
    </xs:sequence>
    <xs:attribute name="name" type="sd:TagType" use="optional" />
    <xs:attribute name="manufacturer" type="xs:string" use="required" />
    <xs:attribute name="model" type="xs:string" use="required" />
    <xs:attribute name="swversion" type="xs:string" use="required" />
    <xs:attribute name="hwversion" type="xs:string" use="optional" />
    <!-- This provides the minor version of this device profile; use '0' for this
version -->
    <xs:attribute name="deviceProfileMinorVersion" type="xs:integer" use="required" />
</xs:complexType>
</xs:schema>

```

Annex B SMA Logic (Normative)

This annex specifies details related to SMA Logic. Programmable logic within the SMA gateway is externally expressed as XML documents in the form of macros, statements, expressions, and conditionals. The logic is intentionally minimal, as it is intended to allow for any needed capabilities with a minimum of specification. An important design goal is to make sure that all of the logic constructs are mirrored directly in the XML structure.

All values used in a macro must be expressed in standard interoperable units. It is the responsibility of the SMA gateway to convert the standard units used in the protocol to any device-specific units used by their particular implementation.

The logic elements and constructs defined in this document should be able to fully support interoperable SMA gateway logic needs.

B.1 Macros and Statements

Macros contain only program statements and conditional logic. A statement can contain one of four elements:

- A property value assignment
- A command to execute a method
- A command to run another macro
- A fixed or random delay

B.2 Assignments

There are three different assignment targets and two different forms of the assignment statement. Assignments can be made to:

- A device property, addressed by its URI property-id

```
<SL:Set property="property-id" value="the-value"/>
```
- A local variable, which is scoped local to a single macro

```
<SL:Set local="i" value="1"/>
```
- A global variable, which is global to the gateway and can be referenced from any macro

```
<SL:Set global="userMode" value="private"/>
```

The value that gets assigned can be specified in one of two forms:

- If it's a literal, as an attribute in the assignment element (as in the examples above)
- Otherwise, the value attribute is omitted, and instead the value is specified as a sub-element expression:

```
<SL:Set property="property-id">
  <SL:Call macro="getTheValue" />
</SL:Set>
```

B.3 Method/Macro Invocation

Methods and macros are invoked using the "Call" element. Within the call, either a "method" or "macro" attribute is required.

Use the "method" attribute to call a method on a device. For this case, the method attribute must be set to the URI of the device method to execute. The parameter names must match the names specified in the device profile.

```
<SL:Call method="method-id" param1="first-param" .../>
```

Methods may or may not return a value. If the method does return a value, it can be used anywhere an expression can be used. If the return value is a boolean, the method call can be used as the expression within a conditional element. If a method which does not return a value is used in a conditional, it always evaluates to false. If it's used in an assignment, it evaluates to an empty string (essentially a "null").

Macros are called using the "macro" tag. All macros return a value (either explicitly using the "Return" element, or implicitly a null/false). Thus all macros are themselves expressions, and can be used in assignments, or if they are boolean expressions, in conditionals.

```
<SL:Call macro="getInfo"/>
```

B.4 Wait Statement

Waits can either be fixed or random within fixed limits. Delays are always specified in milliseconds.

```
<SL:Wait period="2000"/>  
<SL:Wait min="100" max="2000"/>
```

B.5 Local Variable Usage

Some program logic will require the use of local or global variables. These are treated exactly like any other resource (i.e., identified by their id) when they are used in a property assignment or a conditional, but can be dynamically created and used.

For example, a local variable (named "i") can be used to create a for loop:

```
<SL:Set local="i" value="0"/>  
<SL:Until>  
  <SL:Equal local='i' value='5'/>  
  <SL:Do>  
    stuff to do  
    <SL:Add local='i' value='1'/>  
  </SL:Do>  
</SL:Until>
```

B.6 Expressions

Expressions are used both in program logic and in trigger evaluation. When used within conditionals, expressions are required to evaluate to a boolean. Any expression that does not evaluate to a boolean is implicitly "false". Expressions may be composed of boolean (and, or, not), numerical comparisons (less-than, equals, less-or-equal) or

object comparison (e.g., state="on"). Since the numerical and object operators can be combined with the logic operators, there is no need for the opposites (greater-than, name!="this"). Note that all operators are implemented as XML elements, never as attributes or as values.

Simple expressions:

```
<SL:Equal property='property-id' value='a-value'/>
```

This evaluates to true if the property specified by the property id endpoint evaluates to true.

```
<SL:Get property='property-id'/>
```

This evaluates to the current value of the specified property.

Combination expression:

```
<SL:And>
  <SL:Equal property='propertya-id' value='a-value' />
  <SL:Equal property='propertyb-id' value='b-value' />
</SL:And>
```

Evaluates to true if both sub expressions are true.

So, if we want a>5, xzy="united", as long as r=1:

```
<SL:And>
  <SL:Equal property='propertyr-id' value='1' />
  <SL:And property='propertyr-id' value='1'>
    <SL:Not>
      <SL:LessEq property='propertya-id' value='5' />
    </SL:Not>
    <SL:Equal property='propertyxzy-id' value='united' />
  </SL:And>
</SL:And>
```

B.7 Conditionals

Conditionals alter the straight line execution of macros. Conditionals use expressions to determine whether or not to execute their body. The body of a conditional is wrapped in a "Then", "else" or "Do" element (depending on the conditional).

B.8 If-Else

```
<SL:If>
  boolean expression
  <SL:Then>
    stuff to do
  </SL:Then>
  <SL:Else>
    stuff to do
  </SL:Else>
</SL:If>
```

B.9 While

```
<SL:While>
  boolean expression
  <SL:Do>
    stuff to do
  </SL:Do>
</SL:While>
```

B.10 Until

```
<SL:Until>
  boolean expression
  <SL:Do>
    stuff to do
  </SL:Do>
</SL:Until>
```

B.11 Switch

```
<SL:Switch>
  <SL:Case>
    boolean expression 1
    <SL:Do>
      stuff to do
    </SL:Do>
    <SL:Break/>
  </SL:Case>
  <SL:Case>
    boolean expression 2
    <SL:Do>
      stuff to do
    </SL:Do>
    <SL:Break/>
  </SL:Case>
</SL:Switch>
```

B.12 Temporal Expressions

In many cases, time is involved with expressions or events. The SMA gateway MUST recognize the following temporal constructs:

NOW

The current time, time of day, day of week.

DAY-OF-WEEK

This is represented as a string of characters for each day of the week, where the capitalized letter of each day is used as follows:

*S*unday, *M*onday, *T*uesday, *W*ednesday, *th*u*R*sday, *F*riday, *s*Aturday.

Sunday, Monday and Friday would be: "smf"

Tuesday & Thursday: "tr"

Etc.

The day-of-week string interpretation must be case-insensitive.

TIME-OF-DAY

This is a string that specifies, in local time, a time of day in hour, minute and second.

Sample temporal expression:

```
<SL:And>
  <SL:DayEq dow='mtw' value='NOW' />
  <SL:After tod='060000' value='NOW' />
  <SL:Before tod='185959' value='NOW' />
</SL:And>
```

This expression will be true on Mondays, Tuesdays and Wednesdays between 6am and 6:59:59pm, local time.

B.13 Return Values From a Macro

By default, all macros return a null/false value. A macro can explicitly return a value using the "Return" element. The return element does not control program flow (it does not terminate the macro) - it simply sets a return value so that the entire macro call is itself an expression.

```
<SL:Macro name='check-day'>
  <SL:If>
    <SL:Equals dow='t' value='NOW' />
    <SL:Then>
      <SL:Return value=' Belgium' />
    </SL:Then>
    <SL:Else>
      <SL:Return value=' Somewhere else' />
    </SL:Else>
  </SL:If>
</SL:Macro>
```

From the 1969 movie named "If it's Tuesday This Must Be Belgium".

The value returned by a macro can be ignored, or it can be used in one of two ways:

- As an expression in a conditional evaluation (if it returns a boolean)

```
<SL:If>
  <SL:Call macro='check-condition1' />
  <SL:Then>
    do stuff
  </SL:Then>
</SL:If>
```

- As a source for an assignment

```
<SL:Set local="lamp_val">
  <SL:Call macro='get-lamp-value' />
</SL:Set>
```

This second example shows the alternate way to assign values: if there is no "value" attribute in the "Set" element, the Set element must contain a child element that is an expression that evaluates to the value to which the target property gets set.

If more than one return statement is executed within a macro execution, the last one is returned as the macro value.

B.14 Triggers

Triggers are a declarative technique to tie the occurrence of an event to a specified macro. They in effect replace a "main()" program in a traditional procedural program. Triggers respond to events that occur in the system, and allow the protocol to express global rules.

Unlike the boolean expressions in a macro conditional, the expression in a trigger is always interpreted as an event, not a steady state. In other words, triggers respond to changes in conditions, and only call their associated macros once, at the point when their expression evaluates to true. Once a trigger's macro is called, the trigger does not call the macro again until the event that is described by its boolean expression occurs again. Note that exactly what constitutes the event occurring "again" may vary significantly with different implementations, which is one of the reasons that the declarative trigger paradigm is used rather than a more specific definition of what "again" means.

```
<SL:Trigger target='macro-id'>  
    boolean expression  
</SL:Trigger>
```

B.15 Schedules

Schedules are variations on triggers. They are declared like triggers, and like triggers they run a when their "condition" evaluates to true. While in principal schedules can be fully represented by the other SMALogic constructs, the complexity of schedules, combined with the fact that they are likely to be implemented in vastly different ways, argues for some special declarative semantics for schedules.

In order to support the common scheduling paradigm "repeat", the special temporal expression element "Every" is added to the existing "Before". "After" and "DayEq". The semantics of "Every" are different from normal expressions, and the Every element cannot be used in a macro or trigger - only within a schedule.

Appendix I SMA Signaling Examples

This appendix presents examples to illustrate how the SMA Signaling protocol can be used for common tasks. SMA messages are sent using a RESTful web services compliant interface (pkt-sma-1). This interface uses HTTP method requests to initiate communication. Such a HTTP request can either be an event or an instruction. If it is an instruction, the SMA Request contains the following:

- Correlation ID
- Timer T1
- Timer T2

The SMA Response to an SMA Request can either be in the body of the HTTP Response, e.g., 200 OK message, or sent across as an event. If sent across as an event, the correlation ID becomes the Event ID.

The examples that follow provide the HTTP Request containing instructions and events and indicate the SMA Response which can be included either in the HTTP Response or an event. When sent as an event, the correlation ID forms the event ID in the HTTP URI.

I.1 Retrieve a specific property of an SMA device

a. SMA Request

METHOD: HTTP GET

URI: http://<SMA gateway ID>/<resource ID identifies the SMA device's property>

e.g.,

http://SMASMA_gatewayABC.ES1.example.com/LIGHT123/brightness

The HTTP Body will contain the following:

```
<I cid="cid121" t1="100"/>
```

b. Response

The following SMA Response is sent in the HTTP response (200 OK)

200 OK

```
<Ev>
  <nsLight:brightness value="12" min="10" max="20"/>
</Ev>
```

Where nsLight is a pre-configured device profile namespace (e.g., http://www.cablelabs.com/deviceprofile/light)

I.2 To retrieve all properties specific to an SMA device

a. SMA Request

METHOD: HTTP GET

URI: http://<SMA gateway ID>/<resource ID identifies SMA device>

e.g.,

http://SMASMA_gatewayABC.ES1.example.com/LIGHT123

The HTTP Body will contain the following:

```
<I cid="cid121" t1="100"/>
```

b. Response

The following SMA Response is sent in the HTTP response (200 OK)

200 OK

```
<Ev>
  <nsLight:props status="ON">
    <brightness value="12" min="10" max="20"/>
  </nsLight:props>
</Ev>
```

Where nsLight is a pre-configured device profile namespace (e.g., <http://www.cablelabs.com/deviceprofile/light>)

I.3 Retrieving a specific set of properties for an SMA device

a. SMA Request

METHOD: HTTP GET

URI: http://<SMA gateway ID>/<resource ID identifies the subset of properties>

e.g.,

http://SMASMA_gatewayABC.ES1.example.com/LIGHT123/status,brightness

The HTTP Body will contain the following:

```
<I cid="cid121" t1="100"/>
```

b. Response

The following SMA Response is sent in the HTTP response (200 OK)

200 OK

```
<Ev>
  <nsLight:props status="ON">
    <brightness value="12" min="10" max="20"/>
  </nsLight:props>
</Ev>
```

I.4 To retrieve all properties of all SMA devices connected to an SMA gateway

a. SMA Request

METHOD: HTTP GET

URI: http://<SMA gateway ID>/<resource ID identifies all SMA devices>

e.g.,

http://SMASMA_gatewayABC.ES1.example.com/ALLCONNECTEDSMADEVICES

The HTTP Body will contain the following:

```
<I cid="cid122" t1="100" t2="2000"/>
```

b. SMA Response

The following SMA Response is sent as an event

HTTP POST

http://Event Server.example.com/SampleSMASMA_gateway/event/cid122

```
<Rsp>
  <nsLight:Dev id="LIGHT123">
    <props status="ON">
      <brightness value="12" min="10" max="20"/>
    </props>
  </nsLight:Dev>

  <nsMS:Dev id="MS123">
    <props status="OFF" sensitivity="50"/>
  </nsMS:Dev>

  <nsCAM:Dev id="CAM123">
    <props status="ON">
      <XYZ value="12" min="10" max="20"/>
    </props>
  </nsCAM:Dev>

  <nsProgram:Dev id="Prg1">
  <nsLogic:IF> condition 1 </IF>
    <AND/>
  <IF> condition 2 </IF>
  <ACTION>
```

```

    </nsLogic>
  </nsProgram:DEV>
</Rsp>

```

Where some of the namespaces are pre-known to the SMA device

```

nsLight=http://www.cablelabs.com/deviceprofile/light
nsMS=http://www.cablelabs.com/deviceprofile/MS
nsCAM=http://www.cablelabs.com/deviceprofile/CAMERA
nsPROGRAM=http://www.cablelabs.com/deviceprofile/PROGRAM
nsCONTROLLER=http://www.cablelabs.com/deviceprofile/CONTROLLER
nsLOGIC=http://www.cablelabs.com/deviceprofile/LOGIC

```

I.5 Retrieve an SMA gateway's devices, capabilities and macros

a. SMA Request

METHOD: HTTP GET

URI: http://<SMA gateway ID>/<resource ID identifies the SMA gateway>

e.g.,

http://SMASMA_gatewayABC.ES1.example.com/

The HTTP Body contains the following:

```
<I cid="cid121" t1="100"/>
```

b. SMA Response

```
<Rsp>
```

```

<nsSG:Capabilities>
.
.
.
</nsSG:Capabilities>

```

```
<nsSG:Devs>
```

```

  <nsLight:Dev id="LIGHT123" dName="MYOWNLIGHT" />
  <nsMS:Dev id="MS123" dName="MS1" />
  <nsCAM:Dev id="CAM123" dName="YOURCAM" />
</nsSG:Devs>

```

```
<nsSG:Macros>
```

```

.
.
.
</nsSG:Macros>

```

```
</Rsp>
```

I.6 Updating a Property

a. Request

METHOD: HTTP POST

URI: `http://<SMA gateway domain name>/<Resource ID identifies SMA device>/event`

e.g. `http://SMASMA_gatewayXYZ.ES1.example.com/LIGHT123/event`

body:

```
<Ev>
  <I cid="cid121" t1="100">
    <nsLight:Dev id="LIGHT123">
      <props status="OFF">
        </props>
      </nsLight:Dev>
    </p>
  </Ev>
```

b. Response

If successful the SMA gateway returns a 200 OK with the following in the body

```
<R res=true/>
```

If the HTTP Response does not contain the SMA Response then an independent event is transmitted using a HTTP POST message

e.g. `http://SMASMA_gatewayXYZ.ES1.example.com/LIGHT123/event/cid121`

```
<R res=true/>
```

Appendix II SMA Logic Examples

II.1 Example 1

A macro to turn lamp1 on to 50%.

```
<SL:Macro id="pgm-set-50pct">
  <SL:Set type="property" name="uri/lamp1/level" value="50"/>
</SL:Macro>
```

II.1.1 Example 1a

A trigger that runs the above macro when switch1 is set to "1":

```
<SL:Trigger target='pgm-set-50pct' id="trigger1">
  <SL:Equals type="property" name='uri/switch1/state' value='1'/>
</SL:Trigger>
```

II.1.2 Example 1b

A second trigger that calls pgm-set-50pct when switch2 is turned on, but only if the blinds are closed:

```
<SL:Trigger target='pgm-set-50pct' id="trigger2">
  <SL:And>
    <SL:Equals type="property" name='uri/switch2/state' value='1'/>
    <SL:Equals type="property" name='uri/blinds/position' value='down'/>
  </SL:And>
</SL:Trigger>
```

II.2 Example 2

Another way to accomplish example 1b. Here the trigger just checks for the switch state (like example 1a), but the macro checks the blinds:

```
<SL:Macro id="pgm-set-50pct-blinds-down">
  <SL:If>
    <SL:Equals type="property" name='uri/blinds/position' value='down'/>
    <SL:Do>
      <SL:Set type="property" name="uri/lamp1/level" value="50"/>
    </SL:Do>
  </SL:If>
</SL:Macro>

<SL:Trigger target='pgm-set-50pct-blinds-down' id="trigger3">
  <SL:Equals property='uri/switch1/state' value='1'/>
</SL:Trigger>
```

II.3 Example 3

Turn lamp1 off if it's between 11pm and 6am and there's no motion from motion-sensor:

```
<SL:Macro id="lamp1-off-after-11pm" id="macro5">
  <SL:If>
    <SL:And>
      <SL:Equals type="property" name='uri/motion1/state' value='0'/>
```

```

        <SL:Or>
            <SL:After tod='230000' />
            <SL:Before tod='060000' />
        </SL:Or>
    </SL:And>
    <SL:Then>
        <SL:Set type="property" name="uri/lamp1/level" value="0" />
    </SL:Then>
</SL:If>
</SL:Macro>

<SL:Trigger target=' lamp1-off-after-11pm' id="trigger22">
    <SL:Equals type="property" name='uri/motion1/state' value='0' />
</SL:Trigger>

<SL:Trigger target=' lamp1-off-after-11pm'>
    <SL:Equals tod='230001' value='NOW' />
</SL:Trigger>

```

Here we have a macro that turns off lamp1 between 11pm and 6am when the room is empty (no motion). In this example, the macro itself can be run at any time, and it properly checks whether the time and occupancy match its rules, and only turns the light off under the right conditions.

The question is when should this macro be run? In this example, it was decided that it should be run in two cases: 1) at 11pm, so that if the room is empty then it can turn off the light, and 2) whenever the room occupancy changes to empty, so it can check if it's within the time window. Hence there are two triggers defined to run the same macro.

II.4 Example 4

This is an example of using a schedule. Note that the direct children of the Schedule element are implicitly and'ed together - they must all be true.

```

<SL:Schedule id='sched1' target='take-pics'>
    <SL:After tod='050000' />
    <SL:Before tod='18000' />
    <SL:Every period='300' />
    <SL:DayEq dow='mtwrf' />
</SL: Schedule>

```

This calls the take-pics macro every five minutes between 5am and 6pm weekdays. Note that the elements that are direct children of the Schedule element.

II.5 Example 5

Here's example 5 redone with schedules and a global "power-save-mode" variable.

The macro itself just checks a global variable to see if it should turn the lamp off.

```

<SL:Macro id='conditional-lamp-off'>
    <SL:If>
        <SL:Get type='global' name='power-save-mode' />
        <SL:Then>
            <SL:Set type='property' name="uri/lamp1/level" value="0" />
        </SL:Then>
    </SL:If>
</SL:Macro>

```

Then this trigger calls the above macro every time motion stops.

```
<SL:Trigger target=' conditional-lamp-off'>
  <SL:Equals type='property=' name='uri/motion1/state' value='0'/'>
</SL:Trigger>
```

A pair of macros define what should be done at the start and at the end of the period:

```
<SL:Macro id=power-saver-start'>
  <SL:Set type='global' name='power-save-mode' value='true'/'>
  <SL:If>
    <SL:Equals type='property name='uri/motion1/state' value='0'/'>
    <SL:Then>
      <SL:Call macro='conditional-lamp-off'/'>
    </SL:Then>
  </SL:If>
</SL:Macro>

<SL:Macro id=power-saver-stop'>
  <SL:Set type='global name='power-save-mode' value='false'/'>
</SL:Macro>
```

There are two schedules defined, one to start power save mode, another to end it.

```
<SL:Schedule target='power-save-start'>
  <SL: TimeEq tod='230000'/'>
</SL: Schedule>

<SL:Schedule target='power-save-stop'>
  <SL:TimeEq tod='060000'/'>
</SL: Schedule>
```

Appendix III SMA device Profile Usage

The Device Profile XML Schema was presented in A.9. This section illustrates how the device profile can be used by a SMA device manufacturer, and how it can be reported via the SMA Signaling interface.

III.1 Example Light Switch XML Schema

Consider the case of a sample vendor describing a light switch as consisting of a power interface and a dimmable interface. This can be specified using the following example XML Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ls="SampleVendorXYZLightSwitch"
xmlns:sd="urn:cablelabs:packetcable:sma:xsd:v1:deviceprofile"
targetNamespace="SampleVendorXYZLightSwitch" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:import namespace="urn:cablelabs:packetcable:sma:xsd:v1:deviceprofile"
schemaLocation="http://www.cablelabs.com/packetcable/sma/xsd/v1/deviceprofile.xsd"/>

  <xs:element name="lightSwitch">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="sd:Device">
          <xs:sequence>
            <xs:element name="power" type="sd:PowerInterface"/>
            <xs:element name="dimmer" type="sd:DimmableInterface"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A light switch that adheres to the above XML Schema can provide a specific XML instance describing its behavior using the following example XML instance:

```
<?xml version="1.0"?>
<p1:lightSwitch name="Weight Sensor" manufacturer="string" model="string"
swversion="string" hwversion="string" deviceProfileMinorVersion="0"
xmlns:p1="SampleVendorXYZLightSwitch" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="SampleVendorXYZLightSwitch
file:///c:/temp/SampleVendorXYZLightSwitch.xsd">
  <p1:power name="">
    <turnOn/>
    <turnOff/>
    <status read="true" write="false" unit="Unknown"/>
  </p1:power>

  <p1:dimmer name="">
    <turnOnTo>
      <brightness>percent</brightness>
      <ramprate>percent</ramprate>
    </turnOnTo>

    <turnOffTo>
      <brightness>percent</brightness>
      <ramprate>percent</ramprate>
    </turnOffTo>

    <dim>
```

```

        <brightness>percent</brightness>
        <ramprate>percent</ramprate>
    </dim>

    <brighten>

    <brightness>percent</brightness>
    <ramprate>percent</ramprate>

    </brighten>

    <brightnessLevel read="true" write="true" unit="percent" min="0" max="100"/>
    <brightnessPreset read="true" write="true" unit="percent" min="0" max="100"/>
    <rampRatePreset read="true" write="true" unit="second" min="0" max="100"/>

</p1:dimmer>
</p1:lightSwitch>

```

The above can be returned when a specific device is queried.

III.2 Example Device Listing

An SMA gateway can be queried to identify the list of connected, active, or inactive devices. Take for example an SMA gateway that is connected to a camera and two light switches. When queried, the SMA gateway reports this information using the following XML document in the SMA Response.

```

<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2005 rel. 3 U (http://www.altova.com)-->
<deviceList xmlns="urn:cablelabs:packetcable:sma:xsd:v1:devicelisting" xmlns:sd="
urn:cablelabs:packetcable:sma:xsd:v1:devicelisting"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://www.cablelabs.com/packetcable/sma/xsd/v1/devicelisting.xsd ">
  <dev tag="Camera" ns="SampleVendorXYZCamera"
    id="cameraA" name="Front Door Camera"/>

  <dev tag="Light" ns="SampleVendorXYZLightSwitch"
    id="light1" name="Living Room Light"/>

  <dev tag="Light" ns="SampleVendorXYZLightSwitch"
    id="light2" name="Backyard Light"/>
</deviceList>

```

Appendix IV Acknowledgements

CableLabs wishes to thank the PacketCable SMA vendor focus team participants for various contributions and efforts that led to the development of this specification.

Jim Hunter, 4Home

Steve Hughey, Arris Interactive

Chuck Duffy, Cisco Systems

Marc Baum, iControl

Gerry Gutt, iControl

Larry Dodds, Ingrid, Inc.

Eric Chisholm, InnVision Networks

Kevin Kraus, IR Security

Jerry Mahler, Motorola

Bryan Field-Elliot, NextAlarm

Eli Brin, Risco Group

Jim Kitchen, uControl

Wade Cohn, uControl

Michel Kohanim, Universal Devices

Chris Rohrer, Westell.

Special appreciation is extended to the following individuals: Jerry Mahler for QoS and Security sections, and his contributions as the primary editor of this document; Michel Kohanim and Jim Hunter for the SMA Signaling sections; Jim Hunter, Gerry Gutt and Michel Kohanim for the SMA device Profile related efforts. Michel Kohanim, Bryan Field-Elliot and Jim Hunter for the Object Model related contributions; Gerry Gutt and Michel Kohanim for the SMA Logic contributions; Chuck Duffy for media session establishment; Chris Rohrer for error responses; Chris Rohrer and Jim Kitchen for the Backoff and retry requirements; and Steve Hughey for contributions related to QoS.

We also appreciate the review comments and suggestions from David Hancock and Kevin Johns (CableLabs), Gerry Gutt (iControl), Michel Kohanim (Universal Devices) and Jerry Mahler (Motorola).

Sumanth Channabasappa and the PacketCable Architecture Team.