Physical Security Interoperability Alliance
PSIA Specification: Common Metadata/Event Formats and Transports
Version 3.1, Rev. 04
May 12, 2016

Confidential Information for PSIA Use Only

PSIA Common Metadata/Event Management Specification Version 3.1 Rev. 04

Disclaimer

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Without limitation, PSIA disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and PSIA disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

No license, express or implied, by estoppel or otherwise, to any PSIA or PSIA member intellectual property rights is granted herein.

Except that a license is hereby granted by PSIA to copy and reproduce this specification for internal use only.

Contact the Physical Security Interoperability Alliance at info@psialliance.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

Revision History	Description	Date	By
V1.2, Rev 0.1	- Added two new session types based on needs from the Area Ctl Working Group. These had been acknowledged in prior versions of CMEM but not listed in the XSDs. Section 9.8 updated to include flow diagrams for raw TCP and UDP sessions. "metaSessionSupport.xsd" has 2 new session type values available in the "SessionProtocolType:" element/type. - GMCH header now has a formally ordained header extension for stateful transactions. This is done in order to obviate the need for MOH headers for simple appended state information. Additionally, transactional/ACK operations for GMCH formatted data is described in Section 8.3(.1,.2,.4) - List by Section: - *Sections 8.3.1/.2/.4* updated for new GMCH definitions for the Structure/Type values and SAH. - *Section 9* updated to add new 'Data Session Model' as a session type. - *Section 9.1 + Table 9.1.1*: Added new 'SessionFlowType' values.	Sept. 27, 2010	R. Richter

	- New *Section 9.8* added to describe new Raw TCP and UDP session types with session flow diagrams. - *Section 9.9.1* updated comparison table for the various session types. - *Section 10.2.2* updated 'metaSession Support.xsd' for new session flow types and session protocol types. - *Section 10.2.4* added new 'metaSessionTypes' to schema - *Appendix C:* Added new GMCH Structure/Type 'enums' to H file. - Added NEW *Appendix D* with reserved SAH Tag field values.		
Version 1.2 Revision 0.2	- Added a new Session Type for Consumer/Client initiated raw TCP sessions inbound to the Source/Server. Section 9.8.2.. - In Section 10.2.3 we have now added the ability for meta-channels to be 'virtual.' I,e. they can now represent any logical grouping of literal and/or virtual metadata sources. Virtual sources also allow subscripting so that internal sub-channels can be addressed individually, and/or in subsets.	Nov. 10, 2011	R. Richter
Version 1.2 Revision 0.3	- Added Resource Requirements table for all /PSIA/Metadata/… based resources in Section 10.2 and Section 10.3 (,,,/Actions) - Added new Section 10.2.4.1 to split out, and better clarify what was originally all lumped into Section 10.2.4. Now, the /PSIA/Metadata/stream resource is described separately from the /PSIA/Metadata/stream/<id> resource.	Nov. 27, 2011	R. Richter
Version 1.2 Revision 0.4	- Final edits, clean-ups, formatting and typographical changes from review feedback. No technical changes. - FINAL CMEM V1.2 Spec Level.	January 24, 2012	R. Richter
Version 1.2.1 Revision 0.5	- Fixed session type tag in Section 9.8.2 example slide. It now matches the XSD.	Oct. 4, 2012	R.Richter
Version 1.3 Revision 0.1	- Added new Transport Profile requirement tables to Section 9, and affirming cross-reference to Section 10.2.4 - Added new Profile-based requirements definitions to the Metadata resource requirement table in Section 10.	Dec. 12, 2012	R.Richter
Version 1.3 Revision 0.2	- Inserted new Sections 10.2.5/10.2.5.1 to define the new metadata/event 'history' related resources.	Dec. 13, 2012	R.Richter
Version 2.0, Revision 0.3	- Added new Sub-classing/Sub-typing notation to the MIDS definition in Section 7.2. This now allows segmented Class and Type definitions that allow wild-carding. - Corrected typos throughout the spec	Dec. 21, 2012	R.Richter

	- Converted v1.3 to v2.0 due to the jump in functional definition: Session Profile, Resource Profiles, Sub-Classing/Typing.		
Version 2.0, Revision 0.4	- Moved Profile requirements tables, for the resources, from Sections 10.2 and 10.3, to Section 4.1, 'Metadata Resource Overview'. This is a better area for coalescing resource descriptions with the Profile-based requirements for resource implementation(s). - Changed Core profile to require at least one input channel. - Added example to 'Metadata/metadataList' to show use of segmented Class/Type subscriptions.	Dec. 24, 2012	R.Richter
Version 2.0, Revision 0.4a	- In Sections 9.2 and 9.3 added more description over data flow and session mgmt. for datastream HTTP sessions. - Appendix 9: I added a new metadata/event type in the 'Metadata' class called 'endOfMetadata'. This allows sources to indicate when they are closing or re-syncing their data flows. - Added Section 9.9.2 (to Section 9.9) to address overall session and dataflow management items.	Jan. 12, 2013	R.Richter
Version 2.0, Revision 0.4b	- Added additional diagram to Section 9.3/Figure 9.3.1 to explicitly describe 3-party mode Asynchronous Reliable Sessions.	Jan. 30, 2013	R.Richter
Version 2.0, Revision 0.5	- Added new Section 9.9 to describe new set of Asynchronous session types and general operational behavior. Includes new diagram. - Updated Section 9.3 to also reference the rules and descriptions in Section 9.9. - Updated "metaSessionSupport.xsd" in Section 10.2.2. New Asynchronous session types were added to the schema and reference examples were updated to reference "v2.0" of the schema. - Updated Section 10.2.4, for latest version of "metaSessionParns.xsd" with new 'metaSessionPersistence' parameter.	Feb. 20, 2013	R.Richter
Version 2.0, Rev. 0.6	- Added clarification in Section 10.2.4.2 that QSPs and XML bodies are mutually exclusive for setting up sessions.	Mar. 4, 2013	R.Richter
Version 2.0, Rev. 0.6a	- In Section 10.2 added a reference to the PSIA Schemas web site for access to the standalone XSD files.	April 5, 2013	R.Richter
Version 2.0, Revision 0.6b	- Made minor revision to Section 9.10.1 to spec CSEC v1.1 as the session level security requirement for PSIA.	May 5, 2013	R.Richter
Version 2.0, Revision	- Clarified POST as required for all, *except* Core profile.	August 27, 2013	R.Richter

0.7/0.8	- Added explicit version attribute to MetaSessionSupport.xsd and example in Section 10.2.2. - Corrected requirements for Core profile in Section 4.1. - Final v2.0 revision, r0.8	October 9, 2013	
Version 2.0.1, Revision 0.8.1	- Updated/corrected 'metaSessionSupport.xsd' in Section 10.2.2 to be compliant with all XML/XSD validation tools. Version attribute was moved from root element to an attribute in the base type. XSD was updated and the example.	December 7, 2013	R.Richter
Version 2.1 Revision 0.1	- Added Appendix 11 for the definition of "Batttery State Events" based on updates to the v2.1 Service Model specification. - Added clarification for GMCH and raw XML payload definitions in Section. 9.1.2.	May 3, 2014	R.Richter
Version 2.1 Revision 0.2	The XSD contetnts for "Battery State Events" was moved from Appendix 11 to the 'PSIA Service Model v2.1' specification. In Section 9, pp 48-50, the RESTSyncSessionInTargetSend was corrected to RESTSyncSessionTargetSend (Ticket #194)	May 23, 2016	R.Richter
Version 2.1 Revision 0.3	In Sections 9.1.2.1 and 9.2.4 Added support for Core profile use of Native XML payloads.	Aug. 17, 2014	R.Richter
Version 2.1 Revision 0.4	For core access control profile, use RESTSessionSrcIntoRawTCP instead of RESTSessionSrcOutRawTCP per ticket 219. Also fix copy/paste error	December 4, 2014	J Longo
Version 2.1 Revision 0.5	- In section 4 update the index as Optional. - In section 4.1 change profiles with Topologies.	February 11, 2015	Praveen Jha
Version 3.0 Revision 0	- More topology updates… Updated version to 3	February 18, 2015	J Longo
Version 3.0 Revision 1	- TLS requirement is no longer a profile for Raw TCP/UDP – it is part of the streaming spec	February 24, 2015	J Longo
Version 3.0 Revision 2	- Updated examples	February 25, 2015	J Longo
Version 3.0 Revision 3	- Finalized by Group	March 5, 2015	J Longo
Version 3.1 Revision 1	- Removed deprecated (non-implemented functionality). Moved to Deprecated CMEM document - Added Simple Reliable POST Method - Removed schemas where examples were present. Please refer to independent schema files.	October 21,2015	J Longo
Version 3.1	- More removal of deprecated gmch references. More to go though.	November 18,	J Longo

Revision 2	- Additional details for Simple Reliable POST	2015	
Version 3.1 Revision 3	- Reformatted document	January 2016	D Bunzel
Version 3.1 Revision 4	- Finished formatting left to be done from rev 3	May 12 2016	J Longo

Contents

Figures

Tables

1.0 Introduction

This document is the PSIA's architectural specification for a common set of definitions and methods for the processing and management of various forms of metadata information within a cohesive, common PSIA system framework. This document also outlines definitions for event information, in addition to metadata information, since the former is viewed as a subset of the latter. This design philosophy is held based on the fact that all of the aforementioned data types are descriptive, annotative, and correlative with respect to video, and audio, information, whether live or recorded. Since this information is usually not 'standalone', but referential, it is all treated under the same data processing umbrella. In other words, these information types become the notification instances, search criteria, the situation markers, the scene annotations and the time point indicators that give overall reference to a given set of multimedia data. Unifying the processing of metadata and event information also unifies the processes used to manage the various forms of media-related, media qualifying information. This, in turn, enables a common ability to view, record, and search metadata along with the media it relates to.

This document is based on the PSIA Service Model specification. Some, but not all, of the information within the Service Model spec is repeated here in this document for the sake completeness. However, readers and implementers are expected to be familiar with the PSIA Service model and the REST concepts that form the foundation of the PSIA's protocols. Additionally, the streaming of media, and media-related, information is based on the IETF's RTSP/SDP/RTP protocol suite. The information contained in their respective RFCs is referenced within this document, but not repeated.

2.0 Metadata/Event Management Requirements

In order to encapsulate requirements, PSIA has created profiles within each functional specification. A Profile is based off of one and only one PSIA functional specification, however they will identify what PSIA topology they adhere to as well as include an event streaming profile which by nature will identify further requirements from the common specifications, such as CMEM.

The four topologies that can be identified by a profile include:

- Master-Slave over the Internet
- Master-Slave over a LAN
- Peer – Peer over the Internet
- Peer – Peer over a LAN

CMEM identifies required services and resources by topology as well as defining several event streaming profiles that encapsulate methods of event streaming. The implementer need only implement services, resources, and event streaming methods that are identified by the profile and topology.

More information regarding PSIA Profiles and Topologies can be found in the PSIA Service Model specification.

3.0 PSIA REST Overview

REST is an approach to creating services that expose all information as resources in a uniform way. This approach is quite different from the traditional Remote Procedure Call (RPC) mechanism which identifies the functions that an application can call. Put simply, a REST Web application is noun-driven while an RPC Web application is verb-driven. For example, if a Web application were to define an RPC API for user management, it might be written as follows:

```
GET http://webserver/getUserList
GET http://webserver/getUser?userid=100
POST http://webserver/addUser
POST http://webserver/updateUser
GET http://webserver/deleteUser?userid=100
```

On the other hand, a REST API for the same operations would appear as follows:

```
GET http://webserver/users
GET http://webserver/users/user100
POST http://webserver/users
PUT http://webserver/users/user100
DELETE http://webserver/users/user100
```

Part of the simplicity of REST is its uniform interface for operations. Since everything is represented as a resource, create, retrieve, update, and delete (CRUD) operations use the same URI.

3.1 *HTTP Methods and CRUD*

The CRUD operations are defined by the HTTP method as shown in the table below.

HTTP Method	Operation
POST	Create the resource
GET	Retrieve the resource
PUT	Update the resource
DELETE	Delete the resource

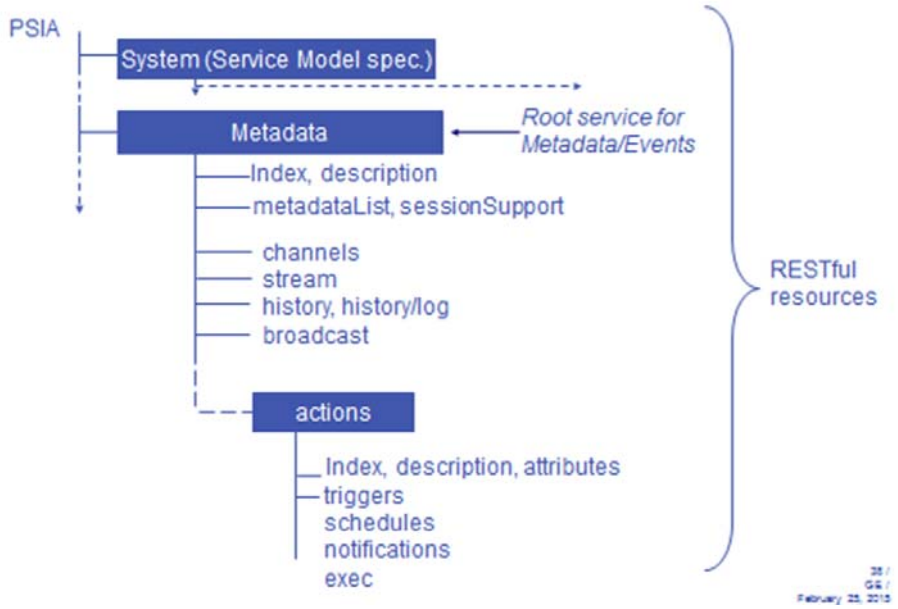
Rules of thumb

- GET calls should never change the system state. They are meant to only return data to the requestor and not to have any side effects
- POST calls should only be used to ADD something that did not already exist.
- PUT calls are expected to update an existing resource but if the resource specified does not already exist, it can be created as well. This will be the assumed default behavior of PUT calls. If any resource

wishes to deviate from this behavior, it should be considered an exception and this should be noted in the implementation notes of the resource.

4.0 PSIA Metadata Resource Structure

Metadata Resource/Service Structure



The above diagram depicts the basic REST resources supported by a spec-compliant PSIA Metadata Service as outlined in this document. The colored boxes in the diagram indicate resources that are PSIA 'Services'.

PSIA Services carry the predefined resources (see the PSIA Service Model specification).

Predefined Resources of a PSIA Core Service	
Resource Name	**Description**
description	Will respond to an HTTP GET with a <ResourceDescription> datablock
capabilities	Will respond to an HTTP GET with a resource-specific datablock
Index	Will respond to an HTTP GET with a <ResourceList> datablock
Indexr	Will respond to an HTTP GET with <ResourceList> datablock

PSIA Services may contain other PSIA Resources, whereas non-Service resources must be "leaf" nodes in the hierarchy. From the PSIA Core Service Model specification: *"Viewed as a tree, services are analogous to branches and resources are analogous to leaves."*
The resource hierarchy determines the REST URI structures used to interact with each resource.

5.0 Overview of Metadata Resources and Implementation Requirements by Topology

Name	Type	Description
Metadata	Service	Base Service resource for all the functional objects within the Metadata Service hierarchy.
index	resource	Required PSIA defined resource that lists the child-level resources within a service.
description	resource	Required PSIA-defined resource that describes the functional attributes of a service/resource.
metadataList	resource	Metadata resource that describes all of the active Metadata/Event types active on a particular device.
sessionSupport	resource	Metadata resource that defines all of the transport, format and session parameters offered by a device for transferring metadata information.
channels	resource	Metadata resource that contains all of the attributes and configuration information for all metadata/event input channels to a device or system.
stream	resource	Metadata resource that acts as the access point for creating metadata/event data streams.
broadcasts	resource	If a source node indicates in its 'sessionSupport' properties that it supports multicast sessions for metadata, then this resource object contains the list of active multicast sessions along with their session attributes.
Actions *(defined in v1.1; see next section)*	Service	Metadata service that provides the ability to query, configure and subscribe to specific actions/notifications offered by a device's metadata service for asynchronous 'push' notification using non-PSIA protocol methods (e.g. Email, FTP, etc.)

The following table defines the PSIA required resource implementations by functional Profile. ***Please note that all profile requirements for resources are independent of the Session Profile requirements listed in Section 11.1.3!***

M-S Internet	M-S Intranet	P-P Internet	P-P Intranet	Resource	Svc/ Re-source	GET	PUT	POST	DELETE
✓	✓	✓	✓	/PSIA/Metadata (base Service)	Svc	N/A	N/A	N/A	N/A
				/PSIA/Metadata/index, /PSIA/Metadata/description	Rsc	✓			
				/PSIA/Metadata/indexr	Rsc	✓			
				/PSIA/Metadata/metadataList	Rsc	✓			
				/PSIA/Metadata/sessionSupport	Rsc	✓			
				/PSIA/Metadata/channels	Rsc	✓			
✓	✓	✓	✓	/PSIA/Metadata/stream	Rsc	✓		✓	✓
				/PSIA/Metadata/stream/<id>	Rsc	✓			✓
				/PSIA/Metadata/clientStream	Rsc			✓	
*	*	*	*	/PSIA/Metadata/history	Rsc	✓			
*	*	*	*	/PSIA/Metadata/history/log	Rsc	✓			
				/PSIA/Metadata/broadcasts	Rsc	✓			
				/PSIA/Metadata/Actions (see ***Section 10.3***)	Svc	**N/A**	N/A	N/A	N/A

*Nodes that are able to maintain an event history ***should*** implement the '/PSIA/Metadata/history' resource along with the ability to stream from its event history log/buffer(s) (see ***Section 10.2.4***).

5.1.1 /PSIA/Metadata/Actions Resource Hierarchy (new for v1.1)

The '/PSIA/Metadata/Actions' Service is optional under all topologies. However, all nodes that support CMEM v1.1, and provide asynchronous notification methods ***should*** implement this service. The 'Actions' service is described, in detail, in ***Section 12.3*** of this document. The table below lists the requirement levels for each 'Actions' resource, when the 'Actions' resource is implemented (i.e. the 'conditional' requirement level).

Name	Type	Description
..Actions/index	resources	PSIA defined resource that lists the child-level resources within a service.
..Actions/description	resource	PSIA-defined resource that describes the functional attributes of a service/resource.
..Actions/attributes	resource	Actions-specific resource that describes functional limits and attributes of the Actions resources.
..Actions/triggers and ..Actions/triggers/<ID>	resources	Definitions of conditions that be classified as events and thereby drive event notifications.
..Actions/notifications and ..Actions/notifications/<ID>	resources	Definitions of specific notification methods that may be employed for certain event triggers.
..Actions/schedules and ../Actions/schedules/<ID>	resources	Week-based calendar definitions that govern when 'triggers' are dormant versus active.

The '/PSIA/Metadata/Actions' service is optional for all topologies.

Resource	GET	PUT	POST	DELETE
/PSIA/Metadata/Actions	N/A	N/A	N/A	N/A
/PSIA/Metadata/Actions/index, /PSIA/Metadata/Actions/description	✓			
/PSIA/Metadata/Actions/indexr	✓			
/PSIA/Metadata//Actions/attributes	✓			
/PSIA//Metadata/Actions/triggers	✓	✓	✓	✓
/PSIA/Metadata/Actions/schedules	✓	✓		
/PSIA/Metadata/Actions/notifications	✓		✓	✓
/PSIA/Metadata/Actions/exec				

5.1.2 Applicability of the Metadata Service Hierarchy

Please note that the above REST resource hierarchy defines the PSIA's ordained Metadata service. This resource hierarchy is ***additional*** to the other PSIA REST services that a device may be required to support to complete its functional definition. For example, a PSIA compliant IP Media Device would implement this Metadata service hierarchy in addition to its System, Streaming, Security, PTZ, and Diagnostics service groups for PSIA compliant support of motion detection events (i.e. it will supplant the current '/Custom/MotionDetection' and '/Custom/Event' resources). PSIA RaCM devices that proxy/forward metadata and events must implement this service in addition to their System, Streaming, Security, ContentMgmt, and other services. As such, much thought has been employed in making the Metadata service hierarchy as simple, and yet, as functional, as possible.

5.2 *Metadata REST URI Examples*

The following examples identify the correlation between the PSIA Metadata resource hierarchy and the REST scheme for addressing these respective resources. Each layer of the Metadata service hierarchy corresponds to a field in a REST URI used to access that resource. The examples below detail this correlation.

- **GET /PSIA/Metadata/metadataList**
  - *Gets the schema instance describing all of the metadata properties for the metadata domain/class/types active on a particular device or system.*
- **GET /PSIA/Metadata/Actions/index**
  - *Returns a schema instance with the metadata event signaling types (denoted by the resources listed) offered by a device. The 'Actions' service is optional.*
- **GET /PSIA/Metadata/sessionSupport**
  - *Returns a schema instance with the supported transport types, formats, and related session parameters supported by a device's Metadata service.*
- **GET /PSIA/Metadata/channels**
  - *Gets the schema instance information that describes each input/source channel of metadata active on a device . If no channel number is provided in the URI, **all** of the channels are listed in the response. Please note that 'channels' is not a resource that provides a data stream; that is accomplished via the 'stream' resource.*
- **PUT /PSIA/Metadata/channels/13**
  - *This transaction updates the configuration of source/input channel #13. A schema instance bearing the configuration information accompanies this message.*
- **POST /PSIA/Metadata/channels**
  - *This transaction creates the configuration of a source/input channel. The ID of the channel, if successfully created, is returned to the creator in the response message. A schema instance containing the required parameters accompanies this message for the creation of a channel.*
- **GET /PSIA/Metadata/broadcasts**
  - *Gets schema instance listing all of the active broadcast session parameters, etc.*
- **GET /PSIA/Metadata/stream**

- *URI for setting up REST-initiated metadata sessions; the associated schema specifies the transport parameters, MIDS and/or channel info.*

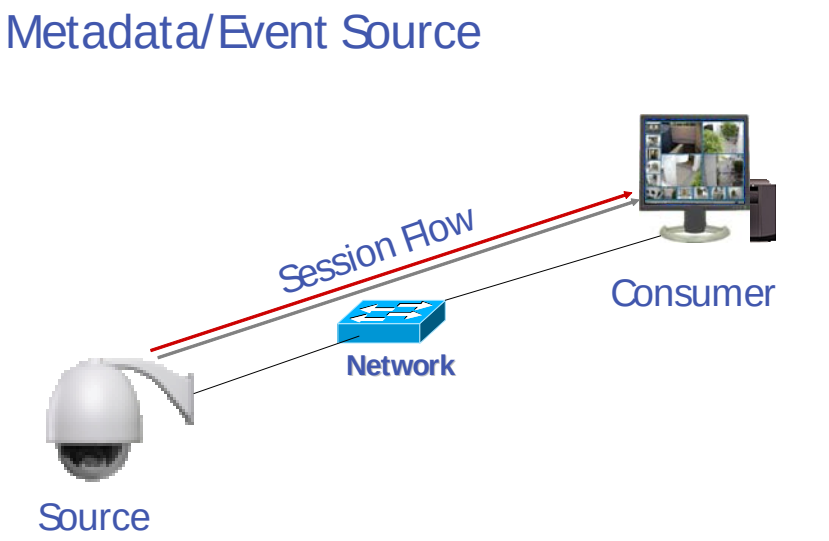
- **POST /PSIA/Metadata/stream/channels/7**
  - *Creates an input/inbound metadata session for metadata on previously created channel #7 based on session parameters supplied in the associated schema instance. This example is only valid for devices that allow inbound 'push' models for receiving metadata/events.*

6.0 Metadata/Event Entities and Roles

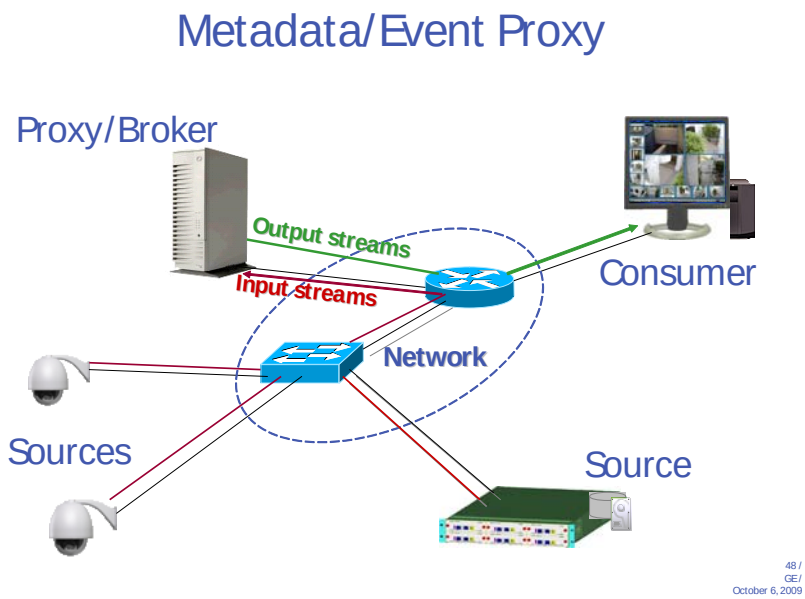
This PSIA specification covers the formats, protocols, and functional behaviors of PSIA compliant devices that deliver metadata and/or events. Operationally, there are 3 primary entities involved in the origination, delivery and consumption of metadata. These entities are:

- **Sources:** Those devices, or systems, that originate metadata and/or event information. This origination may be based on internal processes or attached 'dumb' (i.e. serial based) devices. Either way, the source is the PSIA compliant originator of the information.
- **Proxies:** Proxies receive and forward metadata/event information. Proxies are typically aggregators of metadata/event information and allow consumers to subscribe to specific types of metadata/event information. A common form of a proxy is a PSIA Recording and Content Management (RaCM) device that records audio/video/metadata information from multiple sources and supports sessions to consumers of this information. Proxies are also known as 'brokers'.
- **Consumers:** Consumers subscribe to, and receive, metadata/event information from either sources or proxies.

6.1 ***Simple Metadata Source Access Model***



6.2 ***Proxy/Broker-based Metadata Access Model***



7.0 Metadata Specification Structure

This specification is broken into **four** primary parts. They are listed in order below.

- The **introduction and basic overview**. This consists of the first 5 sections of this document leading up to this point.
- The **architecture overview**. This following part consists of Sections 7, 8, and 9. These sections comprise the architectural and design overview of the PSIA Common Metadata/Event Model (CMEM) which covers data formats, operational behavior, and protocol functionality. Data definitions and protocol/session types are defined in these sections.
- The **interface and data specifications**. Section 10, is the interface and data definition section where the PSIA *Metadata Resource Hierarchy* is described via the data, protocol and interface definitions. This section (Section 10) is the implementation-oriented section where explicit details are specified.
- The '**dictionaries**' of reserved Metadata/Event categories and types are specified in the final appendices of this document.

Readers should have an understanding of the CMEM architecture as defined in the next three sections prior to covering the data definitions specified in Section 10.

8.0 Metadata/Event Architecture Overview

The integration of VMS products with Access Control, Intrusion, Fire and Safety, and Building Automation products drives each of the requirements. Add to these product classes the inclusion and management of Point-of-Sale, ATM, and dry contact state information and you have a picture of the current data types that need to be managed, presently. Even without current standards, the integration activity for merging these many types of disparate information is already underway in the industry. Given this coalescing of product types into multi-domain information systems, the subsequent definitions follow well accepted, and proven, enterprise and network management practices.

8.1 *Common Metadata Format*

The concept of merging metadata and event information into a common data type is based on the fact that both forms of information are descriptive and correlative to multimedia information. In other words, audio/video information is just a 'bunch of bits' without additional information used to add more meaning for searching, extracting, indexing, etc. Consider the following query commands:

"Show all the video clips that coincide with badged entries at the front lobby yesterday between 1:30 – 2:30 PM."

"Find video that has a red car in the main parking lot this morning."

The first query command can be conducted using access control events for badge entries, within a given timespan. From the positive matches of this metadata search, matched time points can be used to index into the corresponding audio and/or video segments. For the second query command, analytics metadata would be used to find scene descriptive information that matches the query criteria. Once the matches have been found, the correlating time points for the matches can be used to index into the associated audio and/or video information. Please note that the above queries assume the VMS software resolves the respective fields-of-view to the appropriate camera(s). Given the fact that both event and metadata information is ultimately used for similar purposes, and the fact that security information is already merging in the industry, it is logical to merge them into a common, but flexible, information class.

The first requirement for uniform management and processing of metadata and event information is the need for a basic, common set of fields that are present in each metadata/event 'atom'. The common fields optimally occupy a fixed area, or ordinal placement, in each 'atom'. Basically, a header that always carries the common data that precedes the other metadata/event-specific information. The recommended common header fields are:

8.1.1 **Table: Common Metadata/Event Fields**

Field	Description
Version	Metadata format version/revision
Metadata ID String (MIDS)	Domain(format)/Class/Type of the corresponding Metadata/Event (see below). Also known in shorthand as "**metaID**."
Source ID	UUID/GUID of the metadata/event source (ISO/IEC 9834-8, ITU X.667)
Source's Local ID (LID)	Channel/stream/track/zone/area/ROI ID of the relevant originating
Time	Absolute time of the generation of the metadata/event info in "xs:dateTime" format
Priority	Needed for differentiating events in a multi-domain environment
Link	Multi-event correlation/reference/linkage ID (UUID/GUID)

For cases where metadata/event information is conveyed in XML format, such as the current PSIA practice, an exemplary XML metadata header would look like the following:

```
<MetadataHeader>
        <MetaVersion>1.0</MetaVersion>
        <MetaID> /psialliance.org /PtOfSale/void/Register9 </MetaID>
        <MetaSourceID>
                {C15768C8-E695-4315-A06E-CF49E1409654}
        </MetaSourceID>
        <MetaSourceLocalID>0</MetaSourceLocalID>
        <MetaTime>2009-03-24T12:29:06.001Z</MetaTime>
        <MetaPriority>4</MetaPriority>
        <MetaLink>0</MetaLink>
</MetadataHeader>
```

The above example references a 'Point of Sale' event. It could reference any metadata or event type. The "MIDS" (see following section) field, referenced by the "MetaID" element name, identifies the metadata/event type via a URI format that guarantees uniqueness via a flexible hierarchical namespace. This is discussed in more detail in the next section (***Section 7.2***). The MIDS is followed by the 'Source ID' ("MetaSourceID") of the event's origin source. All of the source IDs are 128-bit GUIDs such that unique identities are maintained for all nodes. The format of the GUIDs is *compatible* with ISO/IEC 9834-8/ITU X.667(UUIDs). The "MetaSourceLocalID" is the source's local ID (LID) for the channel, stream, zone, area, or whatever other object originated, or is associated with, the metadata information. For most PSIA devices this field is a channel number. If one is not applicable, an ASCII zero ("0") *should* be used. Next, the "MetaTime" field, in 'xs:dateTime' format, lists the origin time of the event. This field is followed by the priority field, "MetaPriority", which lists the associated priority of this event (priorities are covered in ***Section 3.2***). Finally, a link ID, "MetaLink", is listed though it is NULL (NULL = zero) in this example. Since the link ID is NULL it could've been left out of the event information altogether, or terminated as an unpopulated element (e.g. "<MetaLink/>"). Link IDs correlate multiple event occurrences together (i.e. related occurrences share a common link ID). Please note that the required version field is extracted from the 'MetadataHeader' element's version identifier string, "version="1.0"."

The purpose of common 'MetadataHeader' is for this information to be included at the head of all XML schemas/documents that are used for defining metadata and event information. The information specific to a given metadata or event instance follows the MetadataHeader. Only the header information would be required to comply with the proposed format. The following example of a potential Analytics event is given as more complete example.

An example of the inclusion and use of the above XML header definition (type) is outlined in the following hypothetical example (*please reference Section 8.2 for the XSD details*):

```
<SomeAnalyticsEvent version="1.0">
        <MetadataHeader>
                <MetaVersion>1.0</MetaVersion>
                <MetaID>/psialliance.org /VideoAnalytics/Alert/Rule%2023</MetaID>
                <MetaSourceID>{C15768C8-E695-4315-A06E-CF49E1409654}
                        </MetaSourceID>
                <MetaSourceLocalID>99</MetadataSourceLocalID>
                <MetaTime>2009-03-24T12:29:06.001Z</MetaTime>
                <MetaPriority>4</MetaPriority>
                <MetaLink>0</MetaLink>
        </MetadataHeader>
        <AnalyticsEvent>
                        <EventType> alert </EventType>
                        <ID> 12345 </ID>
                        <TimeStamp> 2009-04-10T09:00:00 </TimeStamp>
                        <AlertHeader>
                           <ReferenceID> 67890 </ReferenceID>
                           <Priority> 4 </Priority>
                           <AlertMessage> unusual activity in parking lot
                         </AlertMessage>
                           <Confidence> 0.9 </Confidence>
                    </AlertHeader>
                        <SourceInfo>
                           <Device> {38a52be4-9352-453e-af97-5c3b448652f0}
                         </Device>
                           <ChannelNo> 99 </ChannelNo>
                           <ViewNo> 4 </ViewNo>
                           <VideoSource> {2f1e4fc0-81fd-11da-9156-00036a0f876a}
                         </VideoSource>
                         </SourceInfo>
                         <RuleList>
                           <RuleInfo>
                                   <RuleID> Rule 23 </RuleID>
                                   <RuleName> Red Car Detector </RuleName>
                                   <RuleElementList>
                                           <RuleElement>
                                                   <RuleType> Area </RuleType>
                                                   <Coordinates>
                                                           <Point>
                                                                   <X> 10 </X>
                                                                   <Y> 20 </Y>
                                                           </Point>
                                                           <Point>
                                                                   <X> 600 </X>
                                                                   <Y> 400 </Y>
```

```
                                        </Point>
                                    </Coordinates>
                                     </RuleElement>
                                </RuleElementList>
                                <Action> send PSIA Alert </Action>
                                <RuleDescription> Send alert when red car is found
                               in main parking lot </RuleDescription>
                                <Duration> 10000 </Duration>
                            </RuleInfo>
                        </RuleList>
                        <ObjectList/>
                        <VendorInfo>
                            <VendorName> PSIA </VendorName>
                        </VendorInfo>
                    </AnalyticsEvent>
                </SomeAnalyticsEvent>
```

The use of a standardized header enables common processing, searching, filtering and interrogation of metadata and events without forcing strict, or inflexible, requirements on the subsequent type-specific data.

The above common header fields have been covered, generally. And, most of the fields are obvious as to their function. However, three fields within the header are to be described in more detail. These fields are: A) the MIDS information ID, B) the priority, and C) the link ID. These are covered in the following sections of this document.

8.2 *Metadata Identity String (MIDS; "metaID")*

DATA TYPE: UTF-8 string

In order to have a large variety of metadata types, that can be commonly processed, and yet allow flexibility in designing and developing metadata product components, a hierarchical namespace, forming a metadata taxonomy, is employed. This notation is based on a URI structure. The format is:

/<domain>/<class>/<type>[/attribute/LID][/TransID][/…]

Definitions for the above URI fields are:

8.2.1 **MIDS Field Definitions**

Field/Name	Requirement Level	Comments
Domain	Mandatory	The 'virtual domain' name of the ordaining body for the ***format and definitions*** that are used for the associated metadata/event information. The domain determines the format, and thus the processing and interpretation, of metadata/event instance data.
Class	Mandatory	Domain-specific 'Class' of the metadata/event information. Some examples are: "VideoMotion", "AccessCtl", "PtOfSale", "Intrusion", "VideoAnalytics", etc.
Type	Mandatory	Class-dependent type of metadata/event information. For example, within a class called "VideoMotion" there would be types such as: "motion", "motionStart", "motionStop", "zoneActive", "zoneInactive", etc.
Attribute/LID ('Local ID')	Dependent / Optional	Free-form field that is available for use as additional descriptive information using the following rules: > The convention is that this field MUST be used as the 'Local ID' field for all metadata/event occurrences that are related to, or associated with, a channel/port/stream ID (i.e. the 'source local ID'; see ***Section 7.1***). > For metadata/event occurrences that have no correlation to a channel or port (etc.), this field is optional.
TransID (Transaction ID)	Optional	A string field that uniquely identifies this occurrence instance to the source. If a source entity requires a transactional level acknowledgement, then this field MAY be used as an identifier for expressly acknowledging a specific metadata/event instance. Please note that the source UUID/GUID and timestamp of a metadata/event instance are the standard fields used for uniqueness. Additional fields are optional.

In this hierarchical namespace scheme, the Domain, Class and Type fields are REQUIRED. The Attribute/LID and TransID fields are optional. To provide consistent parsing and decoding, the above described fields are 'positional' within an MIDS URI. Empty slots after the Domain/Class/Type need not be present. Intervening slots that are empty (e.g. an ID field is present but there is no attribute/LID field) are note by adjacent 'slashes' ("//"). The following example depicts an 'empty' URI Attribute/LID field:

`/psialliance.org/Intrusion/alarm//C1EB2D39`

In the above example, a hypothetical intrusion alarm carries a TransID field (“C1EB2D39”) in its MIDS, but no attribute/LID field. As such, the empty attribute/LID field is noted by the adjacent slashes (“//”) after the type field of “alarm.”

Other information may be appended to the end of an MIDS, as needed as long as the defining body publishes their respective field definitions. Any appended information, after the ID field, is ignored by the common/core processing code and considered instance or manufacturer specific.

The figure below depicts the relationship between domains, classes, and types.

8.2.2 **Domain/Class/Type Hierarchy**

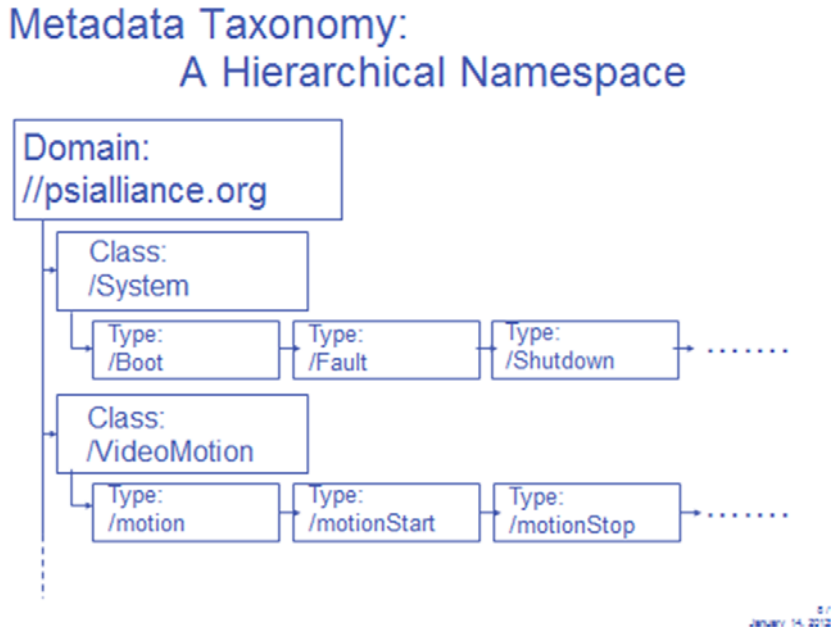


Figure 8-1 Domain/Class/Type Hierarchy

The aforementioned taxonomy enables a vast amount of flexibility in the definition of numerous classes, types, and versions, of metadata information while avoiding ‘collisions’ among metadata publishers. It also advantageously lends itself to subscribing, filtering, and forwarding logic since it is hierarchical in nature with ordinally positioned fields. Additionally, the MIDS design follows well known URI definitions in a REST-like manner, while providing a level of user friendliness via its self-declaring structure. A final benefit is that this structure can be optimized for extremely fast ‘look-ups’ via a technique described elsewhere in this document. The term metadata ‘category’ covers a specific ‘domain/class’ pair.

For procedural definitions, Classes, within a virtual domain, are allocated by the domain authority. In many cases this is the core group, working group or governing committee in a standards body. Typically,

working groups that encompass forms of metadata, are allocated one, or more, pertinent classes as part of their charter. Types within classes are defined and posted for public awareness for all entities by the relevant working groups. Additionally, the allocation of ad-hoc, or vendor specific (i.e. 'roll your own'), is not prohibited by the PSIA. However, any entity publishing its own categories and types, needs to make the PSIA, or the relevant PSIA working group, aware of this activity to prevent confusion and potential definition 'collisions.'

8.2.3 **MIDS Usage**

The hierarchical nature of the MIDS structure is premeditated. As previously noted, it enables large namespaces for metadata and event definitions in a compartmentalized fashion that prevents definition 'collisions' while being flexible and backwards-compatible. Additionally, it lends itself to the processes associated with publishing, or advertising, the data types that comprise the metadata/event definitions governed by PSIA, and other entities. Moreover, the ability to subscribe, describe, and filter metadata/event information is deftly accomplished using its segmented notational syntax.

8.2.3.1 Segmented MIDS Fields (Sub-fields) (New for Versions 1.3, 2.0)

Due to the notation structure of the Domain/Class/Type fields within the MIDs, it is possible to segment a field into more than one segment. For example, a simple Class definition might be "/psialliance.org/Environmentals". However, the defining working group might find it necessary in classifying their MIDS hierarchy to promote additional subfields to allow for simple 'wild-carding' and/or namespace flexibility. For instance the 'Environmentals' Class identifier could be further sub-classed in this manner: "/psialliance.org/Environmentals.internal", "/psialliance.org/Environmentals.external", or "/psialliance.org/Environmentals.regulated". Please note that sub-classing should NOT take the place of the necessary divisions between Classes and their respective Types. For example, using the above hypothetical references, the segmented class "/psialliance.org/Environmentals.internals.humidity" may, or may NOT, be acceptable given the Types that would accompany this Class. I.e. if there were "humidity high" and "humidity low" event types, the defining body needs to consider which of the following MIDS notations is optimal: "/psialliance.org/Environmentals.internal.humidity/high" versus "/psialliance.org/Environmentals.internal/humidity.high" (sub-classing and sub-typing). In this case, the latter is preferable since 'humidity' is typically a subordinate type to the '…Environmentals.internal' Class. And, 'humidity' could also be a Type to the "/psialliance.org/Environmentals.external" Class. It is ***critical*** for each defining body to carefully define their metadata/event MIDS hierarchies with respect to overall function, AND, with respect to the ease and consistency of subscription (see following).

8.2.3.2 MIDS Usage in Subscription/Consumption

Advertising/publishing the support for various metadata/event categories requires, minimally, the domain and class identifiers. Providing the type identifiers is desirable, in many case, but not required. However, for a consumer to specify the criteria for which domain/class/types it desires to receive can also be performed via 'shorthand' notation. Examples follow of how MIDS shorthand notation affects consumption/output.

- "/psialliance.org/IO/active": This MIDS when used as a subscriber notation (see later in this specification, ***Section 10***), indicates the consumer only wants "I/O Active" occurrences.
- "/psialliance.org/Video": This MIDS indicates that the consumer/subscriber wants all of the Video class metadata/event occurrences.
- "//Config": This MIDS indicates that the consumer/subscriber wants all 'Configuration' occurrences irrespective of the domain (a safe bet when all the domains are known). Fundamentally, this notation specifies 'all' domains that have a "Config" class.
- "/psiallance.org/Video//1": This notation indicates interest in all metadata/event occurrences related to 'Video' that occur on input channel #1 (Local ID =1), irrespective of type.
- "/psialliance.org/Environmentals.*": This notation includes a 'wild-card' for a segmented Class (see Section 7.2.2.1 above) called 'Environmentals'. This notation states that the Consumer desires all events related to any Class tag that starts with "/psialliance.org/Environmentals…". For example this would include "/psialliance.org/Environmentals.internal", "/psialliance.org/Environmentals.external" and any other segmented Class tags that start with 'Environmentals…".
- "/psialliance.org/Environmentals.*/humidity.*": This notation, like the above example, covers all 'Environmental…' Classes for all segmented 'humidity' Types such as 'humidity.high', 'humidity.low', or 'humidity.status'.

These examples are provided to highlight and explain how the hierarchical structure of MIDS' are used to help specify subscription requirements.

8.2.3.3 MIDS Extensions

PSIA does not restrict Working Groups from adding fields to any MIDS beyond the currently defined 'domain/class/type/LID/TID' structure defined above. Obviously, any extended fields would need to be 'suffixed' to the end of the currently defined MIDS fields. The only requirement is that the definitions be published by each working group, or manufacturer (for their own domains), in their respective specifications.

8.2.4 **Metadata/Event Uniqueness**

It is significant to note that metadata instances are unique from all other occurrences based on two fields:

- The instance's Source ID, and
- The instance's timestamp.

Though the MIDS indicates 'what' an occurrence is, ***with respect to a specific metadata source, each instance is specifically unique via the timestamp associated with that given occurrence***. Time is discussed in the next section. Please note that the "TransID" is optional information that may be shared between two entities to provide a 'handle' for each event. However, the timestamp for all metadata/event information from a particular source is required to be unique.

8.3 *Time*

DATA TYPE: XML 'dateTime' (W3C format, ISO 8601 compliant)

Every metadata and event instance MUST bear the time of the occurrence. Since the time is one of the two elements used to ensure uniqueness amongst event occurrences, the time granularity of an occurrence, from a specific source, MUST be of such a granularity that no two metadata/event instances, irrespective of category, share the exact same timestamp from the same source. Since there many forms of metadata that may be correlated to video information, the guideline is that all metadata/event occurrences SHOULD have a timestamp that has millisecond granularity (e.g. 2010-02-05T13:27:49.001Z). If a metadata/event source can generate metadata instances at a rate that exceeds the millisecond granularity, additional trailing decimal digits MUST be added to guarantee chronological uniqueness for each instance from that particular source. Please note that the trailing fractional time digits do not have to necessarily reflect perfect sub-millisecond time accuracy – just instance uniqueness. In other words, since most OS's update time on 10 millisecond, or greater, intervals, the least significant digits of a timestamp are not required to be perfectly accurate, but they are required to be unique between to metadata/event instances while being as accurate as is reasonably possible. An example of metadata time management follows.

8.3.1 **Time Management Scenario**

In this example, a hypothetical metadata source has the ability generate event-based metadata information every 412 microseconds. In order to maintain timestamp uniqueness, the source can potentially use either the "GetSystemTimeAsFileTime()" API in Windows (and adjust for the epoch differences between NTP (Jan.1, 1900) epoch and the NTFS time epoch (Jan.1, 1601)), or, if it is Linux based, use the Linux "gettimeofday()" API (and adjust for the UTC epoch versus the NTP epoch). By maintaining an internal NTP-esque 64-bit timestamp, the lower order 21-bits provide a level of granularity that the source can manage to ensure time-level uniqueness per metadata/event occurrence. All of these low order bits are below the millisecond boundary so they can be reasonably

managed without perturbing millisecond level accuracy. For example, a source could use CPU cycle counts to update the low-order time bits, or simply auto-increment the 64-bit timestamp, per event, between OS time updates to keep the timestamp unique. Either method, and potentially others, would be acceptable since devices and systems are not required to keep the timestamp perfectly accurate below millisecond level accuracy; it just has to be unique. Also, it should be noted that sources that do not have occurrence rates at a level that can cause timestamp ambiguity (i.e. at rate greater than the environment's time update capabilities) are not required to keep an internal time level of sub-millisecond accuracy.

8.3.2 **Priorities**

DATA TYPE: Unsigned short int

In a multi-domain system environment, many types of events can occur simultaneously with the potential traffic associated with metadata. For instance, in a system that manages Video, Access Control, Video Analytics, and Point Of Sale information processing, not all information is of the same importance; though it all provides information that correlates to specific instances of audio and/or video data. Additionally, the monitoring of information in a multi-domain system can vary based on the roles and functions of operators, and based on client application types. Just as networks allow the assignment of QoS levels, priorities enable known behavior types for the occurrences of certain events irrespective of other system and metadata activities.

All metadata instances MUST have a priority set in the 'MetaPriority' field. There are 8 priorities ranging from 0 -7 in descending order of priority. The priority levels match the priorities specified by the SysLog facility (RFC 5424). These priority levels are:

8.3.3 **Priority Definitions**

Priority Value	Priority Name	Priority Description
0	Emergency	System or device is unusable; Hard fault/Total failure or catastrophic occurrence.
1	Alert	Action must be taken immediately
2	Critical	Critical condition(s) occurred
3	Error	Error condition(s) occurred
4	Warning	Significant/abnormal/warning conditions have occurred
5	Notice	Normal but noteworthy conditions occurred
6	Informational	Informative messages
7	Debug/Diagnostic	System/Device debug messages

Table 8-1 Priority Definitions

Priorities are assigned to each form (i.e. Class/Type) of metadata as default values by the ordaining standards group, or, in some cases, manufacturer. These default values are to range from 6 (Informational) – 3 (Error) and are shaded in blue in the above table. Values 2 (Critical) through 0 (Emergency) are reserved for user/administrator configuration such that only a user can assign, or promote, a metadata Class and/or Type to one of the highest, actionable levels. Debug (7) information is assumed not occur in normal operation but only under special circumstances (e.g. at the direction of tech support, etc.).

The use of priorities enables systems to process, monitor and/or display metadata/information in chronological, canonical and/or priority level order. This is extremely important when critical events, such as those related to Fire and Safety, etc., are being conjointly processed with other information. Priorities are also very important for ensuring that latencies are removed for important events in latency sensitive environments. Additionally, priorities add another value type for filtering, forwarding and searching.

8.4 ***Link IDs***

DATA TYPE: UUID/GUID (ISO/IEC 9834-8, ITU X.667 format)

Link IDs are UUID/GUID fields supplied for correlating multiple metadata/event instances that are related. For example, a Video Motion Detection (VMD) application detects motion that spans multiple frames of data, or a specific timespan. It sends a 'MotionStart' event followed later by a 'MotionStop' event. The Link ID field allows the source to correlate the two events together as being related by issuing a common UUID/GUID value for the related instances. Additionally, the Link ID enables the ability for users/administrators to retroactively correlate sets of metadata information together (i.e. marking). Since not all forms of metadata information require correlation, this field is OPTIONAL. Also, the use of additional Link IDs, or optional, or domain-specific, fields is allowed in the MetaHeader XSD

via the "xs:any…" extension element. However, inserting additional fields into the required MetaHeader element should only be done judiciously since most optional information should be present in other sections of a schema; only information that is common to all forms of metadata, or are directly related to determining the 'who', 'what', and 'when' aspects of a metadata instance, should be allowed in the MetaHeader.

9.0 Formats, Classification and Multi-Object Metadata/Events

This section of the document deals with the following aspects of metadata and event information processing:

- **Classification:** Efficient, high-speed processing of metadata information that scales with respect to cycles-per-instance (i.e. CPU load per instance). Support for up to 3K instances per second without requiring high-end processor complexes (i.e. quad-core x86 or Xeon class CPUs).
- **Unification:** Uniform processing of all metadata including metadata that includes various forms of information types, and/or is of a different version or format. In addition to this, the ability to meet these requirements with a format that is identical for both messaging and storing metadata information (i.e. no need for transcoding or reformatting).
- **Cohesion:** The ability to include multiple data objects, of identical or disparate formats, per instance in a cohesive manner. Basically this amounts to supporting, when needed, having a base form of metadata (e.g. XML, etc.) and enabling the addition of extra data objects (e.g. images, text, etc.), as needed. This is important for metadata sources that: A) are not equipped to convert or transcode their data into the base format, and B) that include additional information in a format that does not lend itself to conversion into the base format (e.g. image snippets, log files, text from external serial sources, etc.).
- **Automation:** Enabling the processing of metadata such that the creation and implementation of policy engines is greatly simplified for all potential metadata types.

9.1 *PSIA Metadata/Event Information Formats*

Systems and devices in the PSIA protocol framework may select and support Metadata and Event information that is in the following format(s):

- **XML Format**: XML schemas/documents are encouraged as the format for conveying metadata/event information.

Formatted: Line spacing: single, Bulleted + Level: 1 + Aligned at: 0.29" + Tab after: 0.54" + Indent at: 0.54"

The following sections cover these areas regarding the format and required content of the respective formats.

9.2 *XML Metadata/Event Structure*

Systems and devices in a PSIA protocol framework that support Metadata and Event information conveyed in XML format must include the 'MetaHeader" element in each Metadata/Event occurrence being reported. This was outlined in Section 7.1, with an example, above.

MetaHeader.xsd provides the construct for including the named XML type "MetaHeader", using the namespace of "urn:psialliance-org". The "MetaHeader" must be the first element in any metadata/event schema definition. All other information succeeding this header, in any XML schema definition, is implementation dependent.

Please note that the definition of the header allows the extension of the information supplied after the required fields. The "MetaHdrExtension" element allows PSIA Working Groups, and manufacturers participating in the PSIA CMEM framework, to add any pertinent additional information to this header. However, it is recommended that the defining groups should NOT add information to the header unless it is absolutely pertinent; any schema definition including the "metaHeader" type is assumed to have the residual body information defined in its own XSD. One item that may be pertinent as a header extension is the inclusion of additional 'Link IDs' where an event, or metadata, instance may be related to more than one other metadata/event occurrence.

Groups defining header extensions are required to provide a unique identifier string ("ExtensionName") preceding the subsequent header extension information. This is so that a receiving entity can identify who the defining body, or group, is for the format of the remaining extension information. No hard requirements are placed on the structure of the "ExtensionName" field except that it follows a URI structure (i.e. fields are separated via the 'slash' ("/") character), and the first field contains the string tag that identifies the group that owns, and publishes, the format of that particular header extension definition. An example "ExtensionName" would be:

```
/psia.SystemsWG/ExtendedLinkIDs
```

This hypothetical example indicates that the defining group is the PSIA Systems Working Group (WG). Irrespective of the defining/owing group, the definition of all header extensions MUST be published to the PSIA so that the information can be made available to all members.

10.0Metadata/Event Transports

The processing of metadata and events affects the way that these forms of information should be optimally transferred. Additionally, the size of any given system, the network topologies involved, and the amount of metadata transferred within a system, plus the potential requirements for maximum latencies and processing levels, affect the appropriate transport type(s) for the transfer of metadata/events. The following transport types are outlined below:

- **Simple Reliable Get Model:** Consumer/Subscriber opens a session to Source to get metadata/event information. Session lasts as long as the Consumer desires to receive data. Data can happen intermittently or as a stream depending upon the occurrence rate. One socket per consumer. (TCP-based Pull model)
- **Simple Reliable POST Model:** The logical inverse of Simple Reliable Get, the Consumer/Subscriber is expected to be available as a RESTful server and data will be POSTed as necessary from data sources to /PSIA/Metadata/clientStream
- **Data Session Model:** The Consumer requests or establishes a separate connection from the Source using a raw TLS-encrypted TCP or UDP session for transporting normal and/or stateful information in a reliable manner.

Each of the above transport models fills particular needs within given system frameworks. Each model is described in more detail in the following sections.

10.1 *Metadata Session Parameters and Support*

For each of the potential session and transport types that can be employed, a set of parameters is defined such that the supported session types, transport modes, and formats, can be published by metadata sources and/or requested by metadata consumers. These parameters are conveyed and described in 2 XML schemas. These schemas are:

- **Session Support Parameters**: The '/PSIA/Metadata/sessionSupport' resource object contains the 'MetaSessionSupport' schema instance that devices use to advertise their supported attributes for communicating the metadata information.
- **Session Parameters**: When consumers read the session support parameters that a device, or system, supplies, it uses that information to supply the specific, compatible session parameters whenever a session is established. The selected session parameters are passed to the source, from the consumer, in the session parameters schema instance.

The specifics of the above schema definitions are described in more detail below, and in ***Section 10.2*** where the resource hierarchy and schema definitions are contained.

10.1.1 Metadata Session Parameters

The management of metadata sessions is primarily governed by 2 schemas, as mentioned above. The 'metaSessionSupport' and 'metaSessionParms' schemas. The 'metaSessionSupport' schema advertises the session and format parameters supported by a source. Consumers use the 'metaSessionParms' schema to request session establishment with the parameters contained within the accompanying schema instance. These schemas are listed in detail in ***Section 10.2.2*** of this document. However, the element/parameters used in these schemas are described below. The 'Source' column indicates the requirement level of a field within a 'metaSessionSupport' schema instance. The 'Consumer' column lists the requirement level for an element as a session setup parameter.

10.1.2 Transport Information Elements

Element Name	Source	Consumer	Description
'metaFormat'	Required		This element defines which formats are supported for metadata/event transfer. There is currently one choice: • "xml-psia": This represents that the format is the XML/XSD definitions, with the common meta-header information, published by the PSIA working groups. A metadata source MUST advertise which of the above are available when the "/Metadata/sessionSupport" resource is read.
'metaSessionType:: metaSessionProtocol'	Required		This element, within the 'metaSessionTypes' list, describes the session-level transport modes supported by a metadata source, and the transport type requested by a client. Each element contains two fields. The first, "metaSessionProtocol", is required. It describes the session level protocol to be used for transporting metadata information. Consumers may only select the transport modes supported by a source. The transport modes are described in more detail in the following section of this document. The 'metaSessionFlowType' is an optional field. Its use is dependent upon the protocol mode (see above) chosen for transporting
'metaSessionType::	Optional		

metaSessionFlowType'			metadata information. If one of the HTTP/REST based protocols is chosen, then one of the following 2 flow modes must be selected: • "**datastream**": This mode is for 1-way transfer of information, in a data streaming manner, from the source to the consumer. I.e., one 'GET" from the consumer will start a stream of data from the source as metadata is available. This is described in more detail later. • "**transaction**": This flow mode requires the consumer to issue a 'GET' for all data to be received from a source, or in the case of the POST model, the consumer is POSTed to. I.e., a 'GET' is issued for metadata, and once the source provides that data, the consumer issues another 'GET', that acknowledges the received data, to receive the next chunk of metadata information. This GET/response cycle is repeated until the session is to be disconnected. More details are covered later in this specification. •
"metaSessionType":: "metaSessionPersistent	Dependent/ Optional	N/A	This dependent/optional element indicates whether sources can retain asynchronous HTTP/REST session parameters across reboots, or not. The default support level for basic devices is 'False'. However, Proxy devices MUST indicate whether or not they support persistence, and all proxy/broker devices SHOULD support asynch session parameter persistence. See ***Sections 9.3, 10.2.2*** and ***10.2.4***. for details.
'multicastCapable'	Required	N/A	Sources MUST indicate if they support multicast/UDP transmission. This element is irrelevant as a session parameter.
'scheduleCapable'	Required/ Dependent	N/A	Sources MUST indicate if they support the scheduling of asynchronous notification sessions and triggers. CMEM v1.0 nodes must

			indicate FALSE since this feature is reserved for v1.1+. See ***Section 10.2.2*** for more details.
"netAddress" and "netcastMode",	Optional/ Dependent	Optional/ Dependent	These elements are IP (Internet Protocol) addressing and transmission mode parameters. Sources MUST list the IP (net) addresses of multicast sessions that are active. Consumers MUST list IP/net addresses when initiating: A) callback sessions, or B) multicast sessions.
"transactionAck"	Required	Required	This Boolean element indicate whether the Consumer/Source will process data that requires explicit ACKnowedgments. This flag MUST be set to 'True' for both Transactional and Stateful data flow types. Otherwise the source can only send 'stream' data (i.e. non-transactional/stateful) or abort the session.
'metadataNameList'	Dependent	Optional	For sources, this element only has validity for listing the types of metadata associated with multicast, or live input, channels. For consumers, this optional element allows a consumer to apply metadata/event filters to a session such that the consumer will only receive the specific types of data it desires.
'metaChannelList'	N/A	Optional	Consumers may apply a list of input channels for which it desires to receive data from, as part of the session setup parameters. This field, and the above field, can be applied as 'filters' as session setup parameters.

Table 10-1 Transport Information Elements

10.1.3 Metadata Session Requirements by Profile

The following tables explicitly describe the session implementation requirements by the profile definition assigned. Please note that: A) ALL Profile requirements for devices capable of producing events include the Core profile requirement of supporting simple, synchronous HTTP GET sessions, B) nodes may implement more than one profile.

10.1.4 Core Metadata Profile

Session Type	*Format(s)*	*Notes*
RESTSyncSessionTargetSend	Native XML payload with MIME Content type of “application/xml” for single document payloads, or “multipart/mixed” for multi-document payloads.	Synchronous HTTP GET

10.1.5 Low Latency Metadata Profile Option

Session Type	*Format(s)*	*Notes*
RESTSyncSessionTargetSend	Native XML payload with MIME Content type of “application/xml” for single document payloads, or “multipart/mixed” for multi-document payloads.	Synchronous HTTP GET
RESTSessionSrcIntoRawTCP	XML format payload; datastream transfer mode	Raw TCP Session from Event Server to Client

10.1.6 Server Consumer Metadata Profile

Session Type	*Format(s)*	*Notes*
RESTSyncSessionTargetRecv	Native XML payload with MIME Content type of “application/xml” for single document payloads	Synchronous HTTP POST

All of the session types are described in the architectural and design overview in the following section of this specification. Further functional requirements are listed in the /PSIA/Metadata/stream resource definition described in Section 10.2.4.

10.2 *Simple Reliable Get Model (“RESTSyncSessionTargetSend”)*

The Simple Reliable Get Model is a straightforward method for Consumers to get, or harvest, metadata from Sources. It uses the REST session, alone, to initiate the metadata transfers, which makes this method both resource, and firewall, friendly (i.e. if you can get to the source, you’re done). The basic concepts of the Simple Reliable Get Model are depicted below.

10.2.1 Simple Reliable Get Overview

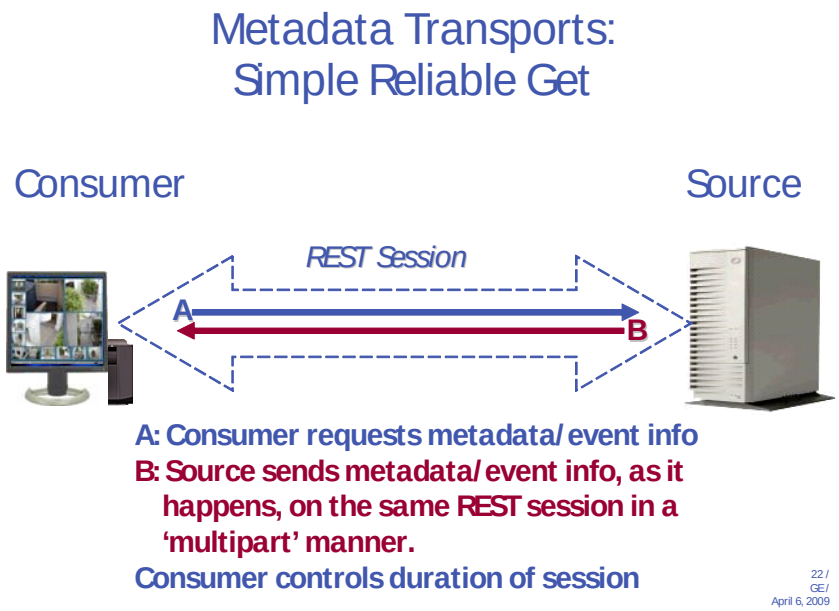


Figure 10-1 Simple Reliable Get Overview

The Simple Reliable GET mode transfers data from the Source to the Consumer based on whether the session was setup in 'stream' or 'transaction' mode (see session parameters). *In stream operation, data is transferred using either HTTP 'multipart' or HTTP 'chunking' formatting since the data stream consists of a set of different objects flowing either continuously or intermittently.*

More detailed exemplary message flows are described below for the Simple Reliable Get Model.

10.2.2 Simple Reliable Get (Message Flow Examples)

The following message flow diagrams cover the 2 format/flow types that are available for HTTP/REST based session types using the Simple Reliable Get transport mechanisms. The first two flows covered are stream based (i.e. "metaSessionFlowType"="datastream"). The subsequent 2 are specific to devices that are transactional (i.e. state based) with respect to metadata/event information exchanges. Please note the subtle differences in the following protocol related items:

Comment [PJ1]: There should be only 2 format/flow types now since we removed GMCH. One example each of datastream and transactional.

- Differences in the format values (i.e. "metaFormat…");
- Differences in the HTTP 'content type', and other, HTTP header values.

Also note that all of the following diagrams use excerpted fields from the “MetaSessionParms” schema indicate the parameters associated with the session initiation. This schema is defined later in this document.

Note, if an unsolicited GET is called to /PSIA/Metadata/stream without first setting up a stream (POST to /PSIA/Metadata/stream), the following parameters should be assumed:

- Format: xml-psia
- Flow-type: datastream
- Session type: RestSyncSessionTargetSend

10.2.3 Simple Reliable Get Sessions in ‘datastream’ Mode

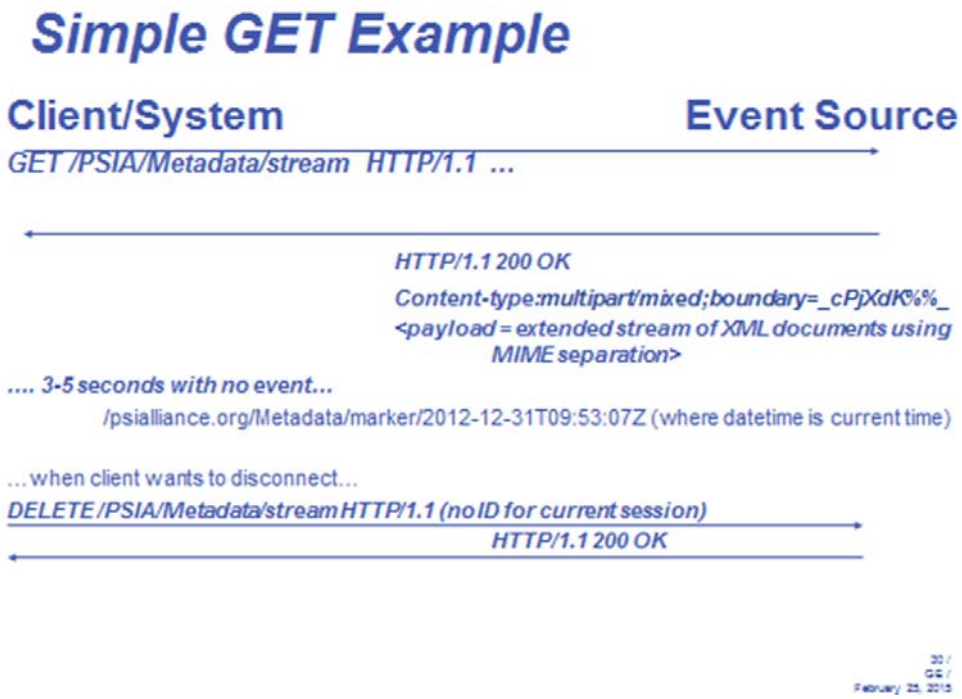


Figure 10-2 Simple Reliable Get Sessions in ‘datastream’ Mode

In the above XML example, the data stream is comprised of XML document instances. Therefore, the content type is “multipart/mixed” with MIME boundary markers for the start/stop of each document. Please see RFCs 2616/2387 for more details.

10.2.3.1 Simple Reliable Get Mode Rules of Operation

All Simple Reliable Get Mode sessions, in 'datastream' mode, are treated as continuous HTTP information streams that may have 'gaps.' These 'gaps' are the time duration segments in a session where data may not be present. Since a simple GET session is intended to be a long-lived session, any gaps of 3-5 seconds without information should be filled, by the source, with a "/psiallliance.org/Metadata/marker…" event. This is the equivalent of a 'keepalive' at the application layer. If a source desires to terminate a Simple Reliable Get session, or desires to terminate the format of data sequence thus requiring the consumer to reissue a 'GET' (i.e. a 'resync'), then a "/psialliance.org/Metadata/endOfMetdata/…' event should be issued by the source, along with the closing multipart boundary marker.
Please note that the "/psialliance.org/Metadata/endOfMetadata/…" event is used for indicating the need for a graceful shutown of the session, or, that the source desires to terminate the multipart object stream and have the consumer re-issue another get; this is called a re-sync. This allows various web implementations to have more control over session management and data flow. More details are covered in. When the source issues an endOfMetadata event, depending on the reason for the endOfMetadata marker (see Appendix 9), the client may be expected to issue a DELETE to close the stream and potentially re-establish the stream.

10.2.4 Simple Reliable Get Examples in Transaction Mode

The next message flow examples cover the transactional (reliable, state oriented) exchange of metadata information. ALL transactional acknowledgements are done via an HTTP response message bearing an XML payload that contains an instance of the "MetaTransactionResponse" schema. For XML formatted exchanges, ALL XML metadata instances are acknowledged using the "dateTime" of the original event data. The following flows cover the basics. However, the details can vary per scenario, so please review the "MetaTransactionResponse" schema definition for details.

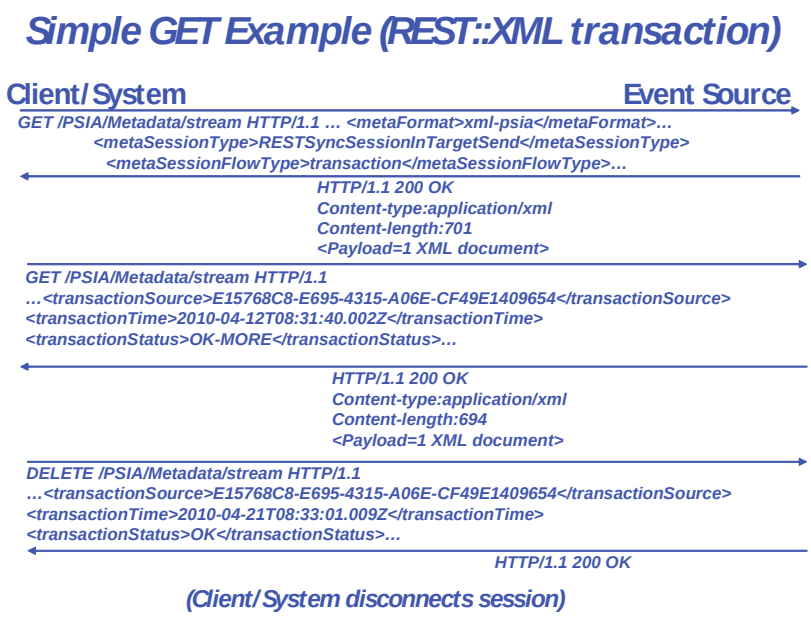


Figure 10-3 Simple Reliable Get XML Transaction

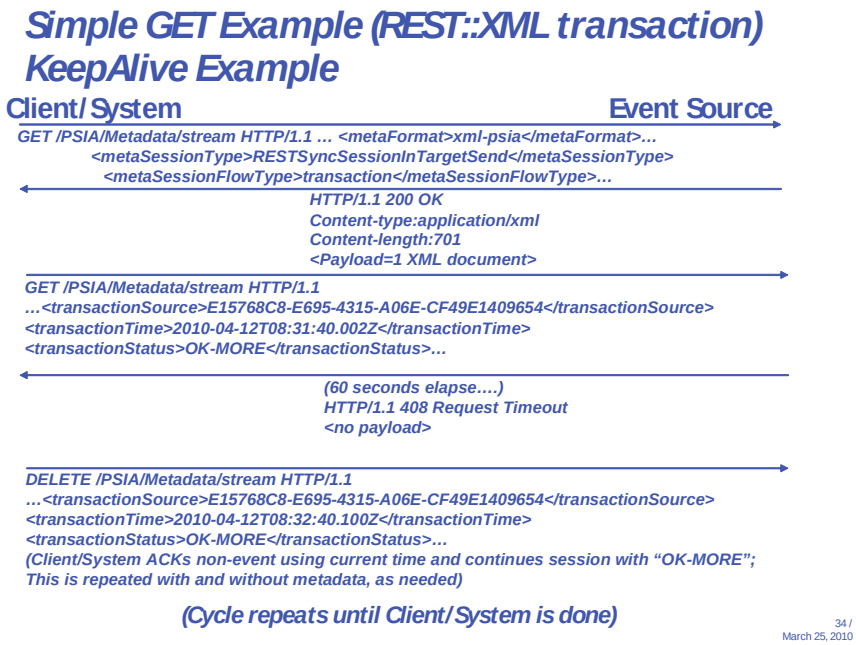


Figure 10-4 Simple Reliable Get XML w/ Keepalive Transaction

The above message flow example references a scenario where the source does not have information to provide for a significant duration (in general, > 30 seconds) while a 'GET' is still outstanding from the consumer. In this case the source sends a response without a payload, and an HTTP status code of "408" indicating it timed-out. This is the equivalent of an application layer 'keepalive' in this session mode. If the consumer desires to continue it issues an acknowledgement, using its own current time (i.e. not an event related time) with a transaction status of "OK-MORE" indicating that it wants the source to continue the active session. This may be repeated as often as is necessary.

10.3 *Simple Reliable POST Model*

In the most basic sense, the Simple Reliable POST model (RESTSyncSessionInTargetRecv) is the reverse of the Simple Reliable GET model. With the Simple Reliable POST model, the REST Server is expected to be the event consumer, with the REST Client "POSTing" batches of events to the server. Like the Simple Reliable Get model, it uses the REST session alone to initiate the metadata transfer which mamkes the method resource and firewall friendly. The basic concepts of the Simple Reliable POST model are depicted below.

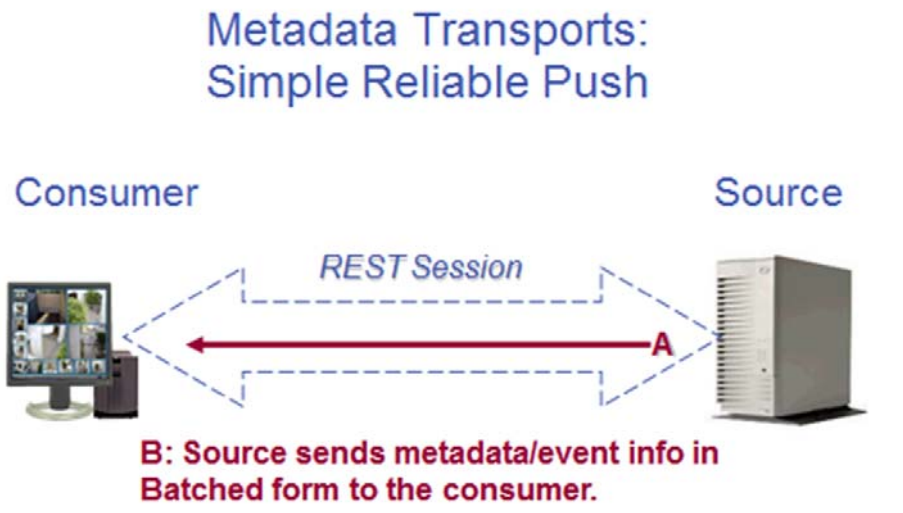


Figure 10-5 Simple Reliable Push

For all Simple Reliable Push transfers, the session is set to 'transaction', as HTTP does not support unbounded POST streaming. As such, the source should have a small buffer in which it collects events and then POSTs all events to the consumer no longer than it would take to send a /Metadata/marker tag – typically within 5 seconds.

Use of Simple Reliable Push is currently governed purely by profile. If the RESTful Server (consumer) advertises a profile that uses Simple Reliable Push, and the RESTful Client supports that profile, then that client, having established communication, will start batching events and POSTing them to the server.

Currently sophisticated flow control is not supplied for Simple Reliable Push. Streaming is expected to begin upon connection establishment, and end upon connection termination. Clients can use HTTP-429 style rate-limiting, or more permanetely try to shut down sources by returning error code HTTP 503. To accept the payload, an HTTP 2xx return code should be returned by the server.

10.4 *Raw Data Session Support ("RESTSessionSrcOutRawTCP", "RESTSessionSrcInRawTCP", "RESTStreamSrcOutRawUDP")*

Support for low-overhead data sessions is provided via the use of XML formatted data over raw TCP (reliable) and UDP (streamed unreliable) connection types. Both of these session types require the use of an HTTP/REST session to setup the actual data connection, as is typical for all PSIA metadata/event session types. The primary differences between the raw TCP and UDP connection types are:

- TCP connections are reliable and, capably, bidirectional. This enables the ability to support streaming data, transactional data or mixed datastream types. If transactional (i.e. acknowledged) data is required, only the TCP sessions are suitable due to reliability and duplex data-flow issues; i.e. there is NO UDP support for 'stateful' or 'transactional' data flow types.
- UDP connections provide a means for low-overhead, low-latency data streaming on a per-message basis. This is due to the fact that UDP does not coalesce MTUs into a common socket buffer like TCP does. Also, since metadata/event streaming does not require an RTP clock for synchronization, this removes superfluous processing and message overhead. Please note, that since UDP is not reliable, and uni-directional, transactional data is NOT suitable for UDP sessions.

Note that the TCP or UDP stream must be TLS encrypted.

10.4.1 Raw TCP Data Session Flow

As depicted in the general session flow diagrams below, the Consumer uses an HTTP/REST session to establish the desired session parameter with the metadata Source. After this, the Consumer must determine if it desires the Source to establish the TCP connection, or if the Consumer wants to establish the connection.

In the first case, where the Source established the TCP connections, the SessionProtocolType is conveyed as "RESTSessionSrcOutRawTCP". This causes the Source to initiate a separate TCP connection to the designated "netAddress." This TCP connection is setup immediately by the Source in order to avoid TCP session timeout errors. TCP connections are inherently duplex thus enabling the support of 2-way communication. This is ideal for applications where low-overhead, reliable, transactional/stateful data needs to be exchanged and acknowledged between a Source and Consumer. The diagrams below depict the interaction between the Consumer and Source.

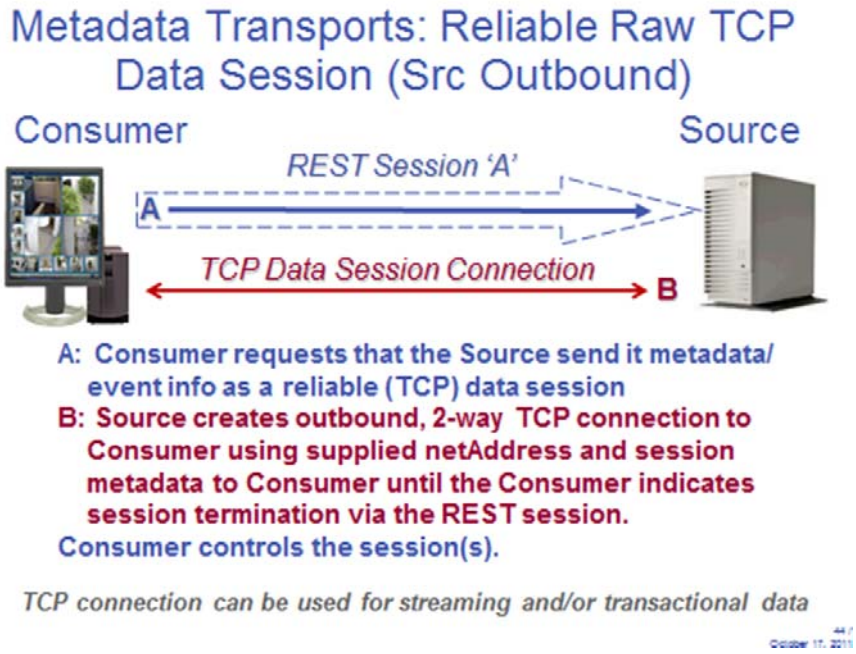


Figure 10-6 Raw TCP Data Session Src Outbound

In the Session level diagram below a Consumer establishes a raw TCP session with a Source via an HTTP/REST control session. This is done via a 'POST' operation to the Source's "/PSIA/Metadata/stream" object. The consumer provides the 3 REQUIRED session parameters for a raw TCP session:

- "metaFormat" set to either "xml-psia" (profile dependent).
- "metaSessionType" MUST be set to "RESTSessionSrcOutRawTCP";
- "netAddress" MUST provide both the IP address AND the TCP port number the Consumer is 'listening' on expecting a connection from the Source; an IP address alone is NOT sufficient for setting-up this connection type;

After successful processing by the Source, a positive HTTP response is sent with a Session ID contained in the PSIA ordained 'ResponseStatus' XML document's "id" field. This session ID is used by the consumer to disconnect/teardown the TCP session, via an explicit 'DELETE' (with the assigned ID) when it is done. If the consumer does not want to have an external HTTP/REST connection active to manage the raw TCP session, it may disconnect the HTTP/REST session and solely manage the TCP connection from its socket interface (i.e. close/shutdown). However, this mode does not enable the Source to determine network/stack errors from managed disconnections.

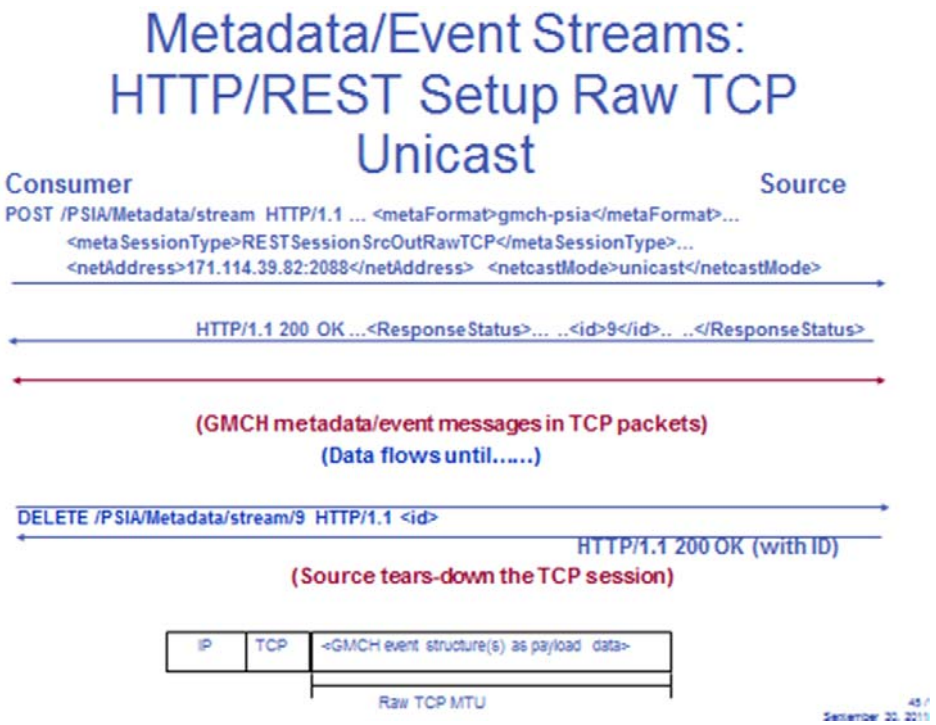


Figure 10-7 Raw TCP Data Session Details

10.4.2 Raw TCP Session Inbound to Source

The ability for a Consumer to establish a raw inbound TCP to the Source (operating as a Server). The setup for these session types is identical to the above setup diagram with the exception that the SessionProtocolType is setup as “RESTSessionSrcInRawTCP”. In the Session parameters the Consumer/Client MUST convey the port address it plans on connecting into, at the Source, as part of the “netAddress.” If the Source can honor the port address, a 200 response code is returned with an “id”. If the Source cannot honor the port address proffered by the Consumer, an HTTP response message with an HTTP status code of “403 Forbidden” with a PSIA ‘ResponseStatus’ payload where the “id” field contains the port number of an acceptable server port. In this case, the Consumer MUST retry the POST operation before establishing the TCP connection to the Source. The diagram below shows the session flow for a standard session.

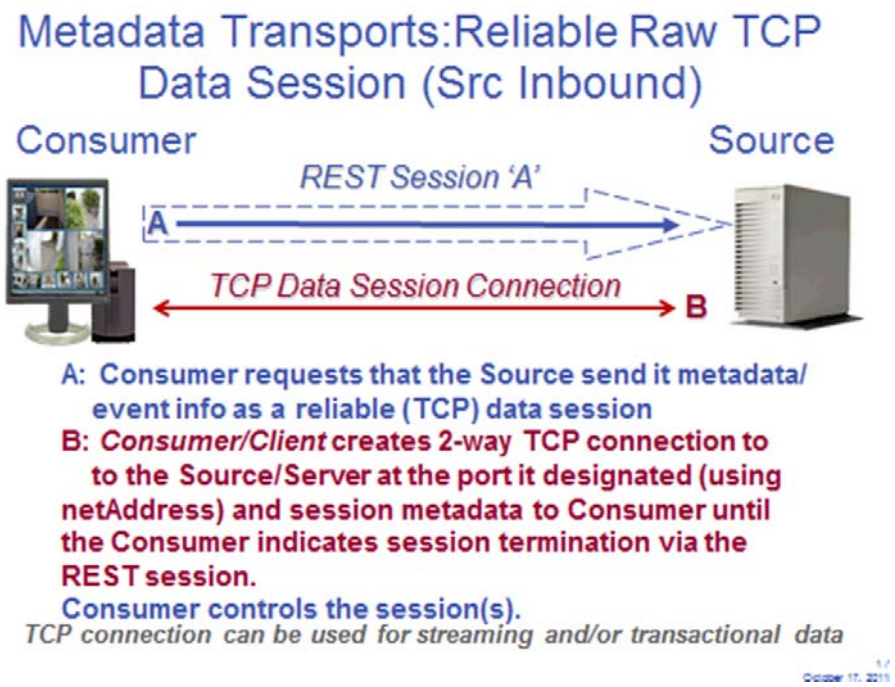


Figure 10-8 Raw TCP Data Session Src Inbound

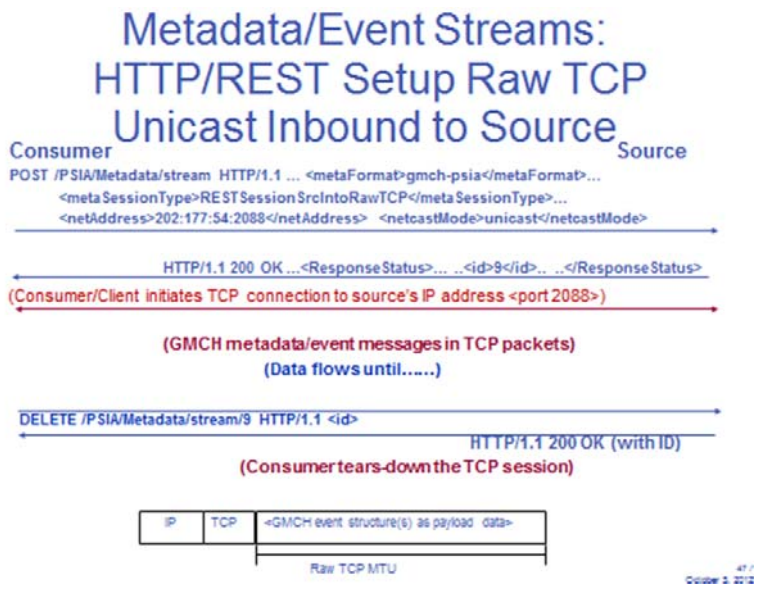


Figure 10-9 Raw TCP Data Session Inbound Details

The above diagrams depict the connection setup, and subsequent data flows, of raw TCP sessions where the Source acts as a 'server' where the Consumer(s) act as clients and establish the TCP data connections inbound to the source. Please note that the mechanism for Source's advertising their 'listening' port is outside the scope of this specification.

10.4.3 Raw UDP Session Flow

Much like the above Raw TCP session description, the UDP version requires the same setup and connection management from the Consumer. The only significant difference between the two connections types, other than the session type parameters, is related to UDP. UDP connections are unidirectional. As such, ***no transactional or stateful data,*** that requires acknowledgement, may be used with this session type. The primary use for raw UDP sessions is for efficient, low overhead reporting of metadata/event information. This session type obviates the need for the overhead of the RTP data stack. A basic data flow diagram for a raw UDP session is provided below.

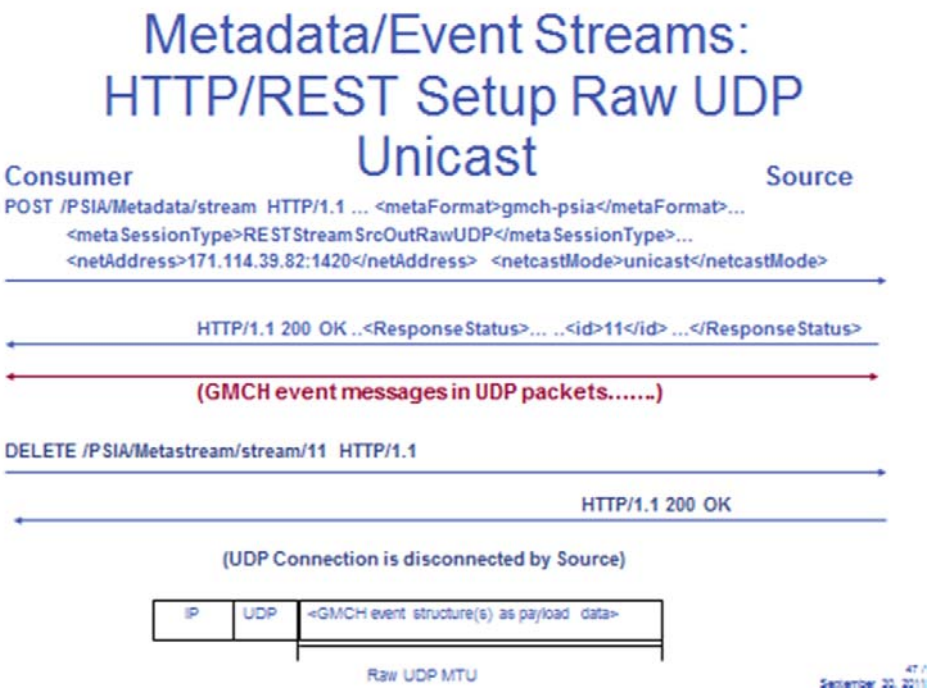


Figure 10-10 Raw UDP Session Details

Unlike the raw TCP connection, a raw UDP session REQUIRES that the Consumer maintains the HTTP/REST control session for the duration of the data session. If the HTTP/REST session disconnects, for whatever reason the corresponding UDP data connection must be closed to prevent 'dangling' sessions.

10.5 *Session/Transport Model Protocol Summary*

The following table provides a protocol level synopsis of the metadata/event transport modes.

10.5.1 Metadata Transport Mode Summary

Mode	Control Protocol	Transport Protocol	Notes
Simple Reliable Get Model	Client REST/HTTP 'Get'	Server REST/HTTP response	Simple, firewall friendly. Can require long-lived sessions or polling.
Simple Reliable POST Model	Client REST/HTTP 'POST'	Server REST/HTTP Response	Simple, firewall friendly. Batched messages can be transferred in one POST.
Raw Data Session Model	Client REST/HTTP	Raw TCP Raw UDP (opt.)	Simple low-overhead connection that allows duplex traffic (TCP), or multicast transmission (UDP). TCP is Firewall friendly.

Table 10-2 Metadata Transport Mode Summary

10.5.2 Session Authentication

As mentioned in prior, HTTP sessions must support authentication as outlined in the PSIA Service Model specification Sections 4.3/4.4, and as defined by RFC 2617. As required by PSIA, this includes support for Digest based authentication. This requirement covers all HTTP sessions irrespective of whether they transport metadata/event information, or just setup the session parameters for other transfer sessions. The scenarios for the setup of asynchronous HTTP/REST sessions is unique. These sessions may require login information to be transferred to the source device as part of the session parameters for the later notification sessions. Any session login information MUST ***not*** be transferred as clear text within a schema instance! If a consumer requires reverse-authentication for asynchronous notification sessions, it must use an HTTPS session to setup the respective parameter information. Session level security requirements may very per customer scenario, however it is highly recommended that devices support TLSv1.x (RFCs 2246/4346) for HTTPS sessions as requirements are defined in PSIA Common Security (CSEC) Model.

10.5.3 General Session Control and Dataflow Management (new v2.0)

All ‘datastream’ mode sessions using HTTP, RTSP/RTP, and raw TCP/UDP transport types require coordination between the consumer and source for efficient session management. The primary areas to be managed are: A) graceful, orderly session shutdown, and B) session inactivity at the application layer. All session data flow examples given in Section 9 are CMEM v1.x compliant. However, CMEM v2.0 (and later) adds some additional mechanisms for managing coordinated session shutdown, and re-establishment, and the proper management of datastream inactivity.

Sources can indicate that they are ending, or re-synching, a datastream mode session by sending a “/psialliance.org/Metadata/endOfMetadata” event. This event class/type has 3 ‘action’ fields that indicate expected action. The action values are:

- “**close**”: The source expects the session initiator to close the session.
- “**resync**”: The source is terminating the HTTP datastream’s data object sequence. This indication requires the consumer to re-issue a GET to retrieve more data. The ‘resync’ is usually issued to close-out multipart object sequence when there may be inactivity.
- “**restart**”: This value indicates that a new session needs to be setup in place of the current one. Some servers have time limits on session life and this indicates a new session needs to be setup for data transfer.

Please note that none of the above metadata/event instances has a payload. They are MIDS only events. The only field accompanying the MIDS is a time stamp successive to the LID field. An example is:

```
“/psilliance.org/Metadata/endOfMetadata/close/2012-12-
   27T09:38:19.71Z”
```

The time stamp is included as marker to indicate the time the source indicated this datastream related event.

Session level inactivity is managed, from the source, by issuing a “/psialliance.org/Metadata/marker” event. This event type acts as a datastream ‘keep-alive’ mechanism. Each ‘marker’ event carries a time stamp just like the above end-of-data events. This keeps the time flow of the data stream synchronized. An example of a marker event is:

```
“psialliance.org/Metadata/marker/2011-09-17T13:44:26.58Z”
```

Metadata sources should send marker event after 3 seconds of data inactivity, unless configured to do so after some other duration. Consumers should note marker events as session status messages. Time stamps are provided for cohesiveness. Events subsequent to a market event must not have a time stamp conflict with a prior marker event (i.e. must be equal or greater)

Transactional mode sessions are not required to use the above mechanisms since they are synchronized on a per-transaction basis. However, v2.0 (and later nodes) may issue an ‘endOfMetadata/resync’ event with a timed-out HTTP request (i.e response status 408).

11.010 Metadata Resource Hierarchy Details

This section of the document covers the implementation of the Metadata service and its resource hierarchy. Each resource in this hierarchy is covered by its 'branch' in the hierarchy. Information from the prior sections covering the architecture are referenced.

11.1 */PSIA/Metadata*

The 'Metadata' base service is the 'root' for Metadata management and streaming in any device that originates, or delivers, metadata, events, and/or alarm information. For IP Media, RaCM, and other PSIA devices, this resource hierarchy is in addition to those required for the other functionality of the device or system.

11.1.1 /PSIA/Metadata/index

This PSIA mandated resource lists all of the 1st level resources contained by '/Metadata'. Please note that the recursive version, 'indexr' is not required. (The 'capabilities' resource description may provide more information.)

URI	/PSIA/Metadata/index		Type	Resource
Function	PSIA Mandatory REST resource/object that enumerates the 1st level child resources for '/Metadata'.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<ResourceList>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes	The 'GET' request issued to retrieve an instance of the 'ResourceList' XML schema. See the Service Model specification, Section 7, 8, 10, for schema details.			

Example

```
<ResourceList version="1.0"
              xmlns:xlink="http://www.w3.org/1999/xlink"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="urn:psialliance-org"
              xsi:schemaLocation="urn:psialliance-org
                                  http://www.psialliance.org/XMLSchemas/service.xsd">
  <!-- See PSIA Service Model specification, Section 10.1.2 -->
  <Resource xlink:href="/Metadata/index">
    <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification, Section
10.1.2), since index is a required resource within a Service -->
    <name>index</name>
    <type>resource</type>
  </Resource>
```

```
  <Resource xlink:href="/Metadata/description">
    <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification, Section
10.1.2), since description is a required resource within a Service -->
    <name>description</name>
    <type>resource</type>
  </Resource>
  <Resource xlink:href="/Metadata/metadataList">
    <name>metadataList</name>
    <type>resource</type>
  </Resource>
  <Resource xlink:href="/Metadata/sessionSupport">
    <!-- PSIA optional resource within a Service -->
    <name>sessionSupport</name>
    <type>resource</type>
  </Resource>
  <Resource xlink:href="/Metadata/channels">
    <name>channels</name>
    <type>resource</type>
   <ResourceList>
   <Resource xlink:href="/Metadata/broadcasts">
      <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification, Section
       10.1.2), since index is a required resource within a Service -->
      <name>broadcasts</name>
      <type>resource</type>
   </Resource>
  <Resource xlink:href="/Metadata/stream">
    <name>stream</name>
    <type>resource</type>
    <ResourceList>
  <Resource xlink:href="/Metadata/Events">
    <name>events</name>
    <type>service</type>
    <ResourceList>
      <Resource xlink:href="/Metadata/Events/index">
        <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification,Section
            10.1.2), since index is a required resource within a Service -->
        <name>index</name>
        <type>resource</type>
      </Resource>
      <Resource xlink:href="/Metadata/Events/description">
        <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification, Section
           10.1.2), since description is a required resource within a Service -->
        <name>description</name>
        <type>resource</type>
      </Resource>
      <Resource xlink:href="/Metadata/Events/triggers">
        <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification, Section
            10.1.2), since index is a required resource within a Service -->
        <name>triggers</name>
        <type>resource</type>
      </Resource>
      <Resource xlink:href="/Metadata/Events/schedule">
        <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification, Section
           10.1.2), since description is a required resource within a Service -->
        <name>schedule</name>
        <type>resource</type>
      </Resource>
      <Resource xlink:href="/Metadata/Events/notification">
        <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification, Section
           10.1.2), since description is a required resource within a Service -->
        <name>schedule</name>
```

```
        <type>resource</type>
      </Resource>
    </ResourceList>
  </Resource>
</ResourceList>
```

11.1.2 /PSIA/Metadata/description

This PSIA mandated resource returns a <ResourceDescription> that describes the functional REST behavior of the Metadata root Service.

URI	/PSIA/Metadata/description		Type	Resource
Function	PSIA REST resource/object that describes the functional behavior of the root Metadata resource (see PSIA Service Model Sections 7, 8, 10 for more details).			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<ResourceDescription>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes	The 'GET' request issued to retrieve an instance of the 'ResourceDescription' XML schema. See the Service Model specification, Section 7, 8, 10, for schema details.			

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ResourceDescription version="1.0"
                     xmlns:xlink="http://www.w3.org/1999/xlink"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xmlns="urn:psialliance-org"
                     xsi:schemaLocation="urn:psialliance-org
                                         http://www.psialliance.org/XMLSchemas/service.xsd">
  <name>Metadata</name>
  <type>service</type>
  <get>
    <queryStringParameters>none</queryStringParameters>
    <inboundXML>none</inboundXML>
    <function>Metadata root service</function>
    <returnResult>ResourceDescription</returnResult>
    <notes>none</notes>
  </get>
  <put></put>
  <post></post>
  <delete></delete>
</ResourceDescription>
```

11.2 */PSIA/Metadata Information Resource Objects*

The resource objects in this section of the specification pertain to the objects that define the categories and operation of metadata/event information active on a particular device or system All of this information is based on a publisher-subscriber model where sources/proxies are publishers and clients/management entities/proxies are all potential subscribers and consumers.
Please note that all of the XSD files listed in the resource description sections below, are accessible on the PSIA schemas web site: "www.psialliance.org/schemas".

11.2.1 /PSIA/Metadata/metadataList

This required resource is the information repository, in list format, for all of the metadata types active on a source. This resource provides consumers, and management applications, a list of all the active metadata types, in 'domain/class', and optionally 'type', format, along with the associated priorities. Optionally, for each class of metadata, the associated input channel can be listed in order to correlate the metadata types with certain inputs. This resource does not allow the creation of metadata types, but it does enable the ability to configure the priority levels associated with each metadata type.

URI	/PSIA/Metadata/metadataList		**Type**	Resource
Function	PSIA REST service that manages the properties/types of supported metadata and metadata channels (if any).			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<MetadataList>	
PUT	N/A	<MetadataUpdate>	<ResponseStatus>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	

Examples

```
<MetadataList version="1.1">
        <numOfEntries>3</numOfEntries>
        <metadataDescrList>
                <metadataDescr>
                        <metaID>
                                <metadataMIDS>/psialliance.org/VideoMotion</metadataMIDS>
                                <metadataPriority>4</metadataPriority>
                        </metaID>
                        <metadataTypeList>
                                <metadataType>
                                        <metadataType>motionStart</metadataType>
                                </metadataType>
                                <metadataType>
                                        <metadataType>motionStop</metadataType>
                                </metadataType>
                                <metadataType>
```

```
                                <metadataType>motion</metadataType>
                        </metadataType>
                <metadataTypeList>
                <metaChannelList>
                        <metaChannel>1</metaChannel>
                </metaChannelList>
        </metadataDescr>
        <metadataDescr>
                <metaID>
                        <metadataMIDS>/psialliance.org/Config</metadataMIDS>
                        <metadataPriority>3</metadataPriority>
                </metaID>
                <metadataTypeList>
                        <metadataType>
                                <metadataType>update</metadataType>
                                <metadataPriority>3</metadataPriority>
                        </metadataType>
                <metadataTypeList>
        </metadataDescr>
        <metadataDescr>
                <metaID>
                        <metadataMIDS>/psialliance.org/System</metadataMIDS>
                        <metadataPriority>3</metadataPriority>
                </metaID>
                <metadataTypeList>
                        <metadataType>
                                <metadataType>boot</metadataType>
                                <metadataPriority>4</metadataPriority>
                        </metadataType>
                        <metadataType>
                                <metadataType>fault</metadataType>
                                <metadataPriority>3</metadataPriority>
                        </metadataType>
                        <metadataType>
                                <metadataType>tamperAlarm</metadataType>
                                <metadataPriority>5</metadataPriority>
                                <metadataItemMode>read-only</metadataItemMode>
                        </metadataType>
                <metadataTypeList>
        </metadataDescr>
    </metadataDescrList>
</MetadataList>
```

In the above example, a simple IP Media device lists that it suppports 3 PSIA-based Classes of metadata: A) VideoMotion, B) Config, and C) System. The VideoMotion class has 3 ‘types’ of metadata in its class: A) motionStart, B) motionStop, and C) motion. All of the VideoMotion class metadata currently has a priority level of 4 since the subordinate ‘types’ inherit the priority of the ‘class’. In addition to the VideoMotion metadata/events, the device also has configuration (‘Config’) metadata/events with the only ‘type’ being ‘update’ (e.g. ‘/psialliance.org/Config/update’) notifications. In addition to these metadata types the device also provides notifications for the System metadata/events ‘boot-up’ and ‘error’; in other words the device provides event notification, in metadata format, for system reboots and critical system-related errors. Please note that the VideoMotion metadata is associated with input channel #1, whereas configuration and system metadata/events are not tied to a specific channel; they are device-wide occurrences.

```
<MetadataList version=”1.1”>
```

```
        <numOfEntries>3</numOfEntries>
        <metadataDescrList>
                <metadataDescr>
                        <metaID>
                                <metadataMIDS>/psialliance.org/AreaControl.*</metadataMIDS>
                                <metadataPriority>3</metadataPriority>
                        </metaID>
                </metadataDescr>
                <metadataDescr>
                        <metaID>
                                <metadataMIDS>/psialliance.org/Environmentals.*</metadataMIDS>
                                <metadataPriority>4</metadataPriority>
                        </metaID>
                        <metadataTypeList>
                                <metadataType>
                                        <metadataType>temperature.high</metadataType>
                                        <metadataPriority>3</metadataPriority>
                                </metadataType>
                                <metadataType>
                                        <metadataType>humidity.*</metadataType>
                                        <metadataPriority>4</metadataPriority>
                                </metadataType>

                        <metadataTypeList>
                </metadataDescr>
                <metadataDescr>
                        <metaID>
                                <metadataMIDS>/psialliance.org/System</metadataMIDS>
                                <metadataPriority>3</metadataPriority>
                        </metaID>
                        <metadataTypeList>
                                <metadataType>
                                        <metadataType>boot</metadataType>
                                        <metadataPriority>4</metadataPriority>
                                </metadataType>
                        <metadataTypeList>
                </metadataDescr>
        </metadataDescrList>
</MetadataList>
```

In the above example, a consumer is subscribing to A) all Area Control events (via a segmented Class with a wild-card '*'), B) Environmental temperature-high events, plus all humidity events (via a segmented Type with a wild-card '*'). <Please see Section 7.2. for notes on segmented Class/Type notation> In addition to these metadata types the device also provides notifications for the System metadata/events 'boot-up' and 'error'; in other words the device provides event notification, in metadata format, for system reboots and critical system-related errors. Please note that the VideoMotion metadata is associated with input channel #1, whereas configuration and system metadata/events are not tied to a specific channel; they are device-wide occurrences.

```
<MetadataList version="1.0" xmlns="urn:psialliance-org">
        <numOfEntries>5</numOfEntries>
        <metadataDescrList>
                <metadataDescr>
                        <metaID>
                                <metadataMIDS>/psialliance.org/VideoMotion</metadataMIDS>
```

```
                        <metadataPriority>4</metadataPriority>
                </metaID>
                <metaChannelList>
                        <metaChannel>1</metaChannel>
                        <metaChannel>2</metaChannel>
                        <metaChannel>3</metaChannel>
                        <metaChannel>4</metaChannel>
                        <metaChannel>5</metaChannel>
                        <metaChannel>6</metaChannel>
                        <metaChannel>7</metaChannel>
                        <metaChannel>8</metaChannel>
                </metaChannelList>
        </metadataDescr>
        <metadataDescr>
                <metaID>
                        <metadataMIDS>/psialliance.org/Video</metadataMIDS>
                        <metadataPriority>3</metadataPriority>
                </metaID>
                <metadataTypeList>
                        <metadataType>
                                <metadataType>signalActive</metadataType>
                        </metadataType>
                        <metadataType>
                                <metadataType>signalInactive</metadataType>
                                <metadataPriority>4</metadataPriority>
                        </metadataType>
                        <metadataType>
                                <metadataType>signalError</metadataType>
                                <metadataPriority>4</metadataPriority>
                        </metadataType>
                <metadataTypeList>
                <metaChannelList>
                        <metaChannel>1</metaChannel>
                        <metaChannel>2</metaChannel>
                        <metaChannel>3</metaChannel>
                        <metaChannel>4</metaChannel>
                        <metaChannel>5</metaChannel>
                        <metaChannel>6</metaChannel>
                        <metaChannel>7</metaChannel>
                        <metaChannel>8</metaChannel>
                </metaChannelList>
        </metadataDescr>
        <metadataDescr>
                <metaID>
                        <metadataMIDS>/psialliance.org/Config</metadataMIDS>
                        <metadataPriority>3</metadataPriority>
                </metaID>
                <metadataTypeList>
                        <metadataType>
                                <metadataType>update</metadataType>
                        </metadataType>
                <metadataTypeList>
        </metadataDescr>
```

```
            <metadataDescr>
                    <metaID>
                            <metadataType>/psialliance.org/System</metadataType>
                            <metadataPriority>3</metadataPriority>
                    </metaID>
                    <metadataTypeList>
                            <metadataType>
                                    <metadataType>boot</metadataType>
                            </metadataType>
                            <metadataType>
                                    <metadataType>fault</metadataType>
                            </metadataType>
                    <metadataTypeList>
            </metadataDescr>
            <metadataDescr>
                    <metaID>
                            <metadataMIDS>/psialliance.org/Storage</metadataMIDS>
                            <metadataPriority>3</metadataPriority>
                    </metaID>
                    <metadataTypeList>
                            <metadataType>
                                    <metadataType>mediaError</metadataType>
                            </metadataType>
                            <metadataType>
                                    <metadataType>mediaFailure</metadataType>
                            </metadataType>
                    <metadataTypeList>
            </metadataDescr>
        </metadataDescrList>
</MetadataList>
```

The above example is based on an 8-port RaCM device (a DVR or NVR). A DVR is a metadata source, whereas an NVR can be both a proxy, for IP camera/encoder inputs, and a source for its own internal metadata/events. In the above, the 8-port device supports both ‘VideoMotion’ and ‘Video’ metadata/events on all 8 channels (i.e. channels 1-8). Since it may be a proxy, the device does not list the ‘Types’ that are active under the ‘VideoMotion’ category. However, it does list the ‘Types’ supported under the ‘Video’ metadata category. The other metadaya classes ‘Config’, ‘System’ and ‘Storage’ are not channel related; they are internal events. The ‘Storage’ class does not give ‘channels’ since these IDs are not particularly relevant to storage media. However, if a ‘Storage’ event occurred, the ‘atrribute’ field in the MIDS should contain a meaningful ID corresponding to the media related to the event.

‘MetadataUpdate’ Example

```
<MetadataUpdate version=”1.1”>
        <numOfEntries>2</numOfEntries>
        <metaDescr>
                <metaID>
                        <metadataMIDS>/metatdata.psia.org/Config/updzte</metadataNIDS>
                        <metadataPriority>2</metadataPriority>
                </metaID>
        </metaDescr>
        <metadataDescr>
                <metaID>
                        <metadataMIDS>/psialliance.org/System</metadataMIDS>
                        <metadataPriority>2</metadataPriority>
```

```
            </metaID>
        </metadataDescr>
</MetadataUpdate>

The above example is based on the prior example 'MetadataList' for a DVR/NVR. The application is
modifying the priority level for two metadata items. This first update item is for Config update events.
The application wants to set the priority level to 2. Please note that the application is using a
shortand notation my supplying a metaID (MIDS) that addresses a specific Class/Type. This can only be
done if the source advertises Types within the metadata classes (see above MetadataList Example #2). The
second item indicates that the application wants to change the priority level of all 'System' metadata to
2 (from 3). This update is applied to the entire 'System' metadata class unlike the prior item.
```

The 'metadataList' resource is the reporting entity for the classes and types of metadata supported, or that are active, on a system or device. This resource is very important since metadata and events are usually consumed based on category, and in some cases, filtered, or restricted, based on sources (usually imposed on Proxy devices). This resource returns a 'MetadataList' document that contains a list of the metadata and event types, by category, in an XML list format. The document elements are:

Element Name	Requirement Level	Notes/Description
"numOfEntries"	Required	Identifies the number of list elements that follow
"metadataDescrList"	Required	List containing "metadata descriptors" (see below)
"metadataDescriptor"	Required	The basic list element, for each metadata domain/class/type that describes the parameters associated with each metadata/event category. The elements in each 'metadataDescriptor' follow:…
"metadataDescriptor:: metaID" (type="metadataIDDescr")	Required	The domain/class, in the PSIA URI format, that describes an active metadata category on a source, or proxy, ***and*** the associated priority level assigned to this metadata/event category. Please note that the generic type used for this element does not require the priority level, since it is used in many places. However, each domain/class (i.e. a 'category') MUST have an overarching priority assigned to it. The use of priorities for 'Types' within categories is optional.
"metadataDescriptor:: metaTypeList::metadataType" (type="metadataTypeDescr")	Optional	This recommended list element describes the 'types', within the above 'domain/class' metadata category, that are active. Sources SHOULD list their types. Proxies may not know the types they are managing per category and are not required to advertise the active types. E.g. a device that advertises a 'VideoMotion'

		category SHOULD also list the types, such as 'motionStart, 'motionStop, etc., that it supports within that metadata/event category. • Each list element is of the "metadataIDDescr" type just like the above "metaID". However, only the 'Type' string needs to be listed. • Priorities are optional for specific 'Types' within a category since categories are required to have an assigned priority value.
"metaChannelList" and "metaChannel"	Required	This required element is a list of the channels (i.e. inputs) that are associated with the ad-vertised metadata/event categories. Each list element ("metaChannel") contains the channel ID of the source channel associated with that specific metadata/event category. This information is provided for scenarios where consumers desire to subscribe to metadata from specific sources.

The above fields are all described in detail in the "MetadataList" XML schema descriptor. This information is retrieved by consumers that perform "GET's" against the '/PSIA/Metadata/metadataList' resource object. This information advertises the categories of metadata/event information that are active on a source or proxy device. The list provides elements that describe the metadata/event category, the priority level for that category, any 'Types' that are active within that category, and a list of input channels that are the origins for that information. Basically, this information is provided such that subscribers can access the information by:

- **Metadata category**: Using the supplied list information, a subscriber can 'GET' information by specifying the categories advertised by a source or proxy.
- **Source**: Since the metadata is listed along with its correlated input channels, subscribers can access the metadata by specifying the desired 'channels' (i.e. sources).
- **Both**: Using combinations of the above, a subscriber can shape the content and sources it desires to gather from a node.

Please note that subscribing to metadata/event information uses the above descriptions but is performed against "/PSIA/Metadata/stream" resource object which is described later in this specification.

11.2.1.1 Updating Metadata Information

Management entities (i.e. users/clients with the appropriate permissions) may change the properties of metadata information, or add input sources to proxies, by performing updates directly to the "/PSIA/Metadata/metadataList" object using an XML document instance compliant with the "MetadataUpdate" schema definition (see above). Using this resource object, and schema definition,

users/clients may modify the priority levels associated with specific metadata/event categories, or even particular domain/class/types should a device list them.

11.2.2 /PSIA/Metadata/sessionSupport

This Metadata resource describes all of the session and format related parameters supported by a PSIA compliant Metadata device or system.

URI	/PSIA/Metadata/sessionSupport		Type	Resource
Function	.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<MetaSessionSupport>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes				

Example(s)

```
<?xml version= »1.0» encoding= »UTF-8 » ?>
<MetaSessionSupport xmlns=urn:psialliance-org version="2.0">
        <metaFormats>
                <metaFormat>gmch-psia</metaFormat>
                <metaFormat>xml-psia</metaFormat>
        </metaFormats>
        <metaSessionTypes>
                <metaSessionType>
                        <metaSessionProtocol>RESTSyncSessionInTargetSend</metaSessionProtocol>
                        <metaSessionFlowType>datastream</metaSessionFlowType>
                </metaSessionType>
                <metaSessionType>
                        <metaSessionProtocol>RESTAsyncSessionOutSourceSend</metaSessionProtocol>
                        <metaSessionFlowType>datastream</metaSessionFlowType>
                </metaSessionType>
        </metaSessionTypes>
        <version>2.0</version>
</MetaSessionSupport>

The above example is for a PSIA compliant basic device. It offers its metadata in either the deprecated
GMCH encapsulated format (not discussed in this document), or as XML documents, in streaming mode.
Additionally, the device complies with the required network session types that it supports.
```

This REST resource advertises the session types, and accompanying formats, that a source or proxy device uses to provide metadata and event information. This is a read-only resource(i.e. only 'GET's are

supported) and cannot be modified. Via this resource a device or system advertises the following session and format related properties.

Element	Notes
"metaFormats" (and "metaFormatList":: "metaFormat")	An element comprised of a list of the formats available for consumption. The choices are: • **"xml-psia"**: XML format for metadata/ event information as published by a PSIA working group, or, in some cases, an external organization that complies with the PSIA CMEM standard.
"metaSessionTypes":: "sessionTypeList":: "sessionType":: "metaSessionProtocol"	An element comprised of a list that describes the session types offered by a device or system for transferring metadata/event information. Each list element is comprised of 3 primary components. The first element, 'metaSessionProtocol' is required for each supported session type. are: A) the session protocol that is supported, and… The individual session protocol tags are described in more detail in this section below. The flow type tags are described in the following table of this document. Descriptions of the session types themselves, are covered in ***Section 9***.
"metaSessionTypes":: "metaSessionTypeList":: metaSessionFlowType"	This dependent/optional element defines the session 'flow types' supported by a device, or node. Since the default support is for 'datastream', devices supporting only this flow type do not need to have this element present. However, event driven devices, and systems, may support 'transaction' mode. Optionally, devices may support both (indicated by 'streamOrTransaction') modes. If a device supports more than the default mode, it MUST supply this element and indicate the flow types supported.
"metaSessionTypes":: "metaSessionTypeList":: "metaSessionPersistent"	Support for Asynchronous HTTP/REST sessions requires that the device indicate if it supports 'persistence', for asynch session parameters, or not. Since the default support value is 'False', this element is not required unless a node supports session parameter persistence across reboots. Please note that Proxy/broker devices/nodes MUST indicate whether they support persistence or not. The recommendation for Proxies is that that SHOULD support persistence. In either case, Proxies must indicate their support level.
"multicastCapable"	This field is required to be present since it indicates whether a device supports UDP multicast transmission of metadata./event information. This form of transmission is optional for endpoints, and recommended for event proxies/brokers.
"scheduleCapable"	This field is required to be present. It indicates, as a Boolean, whether, or not, a device supports scheduling for asynchronous notification

	methods and triggers. Async notification methods include those session types with the tags starting "RESTAsynch…" (see below), the use of non HTTP/REST asynchronous notification session types such as Email/SMTP, FTP, etc., and, (optionally) Multicast/UDP sessions. All v1.0 CMEM nodes MUST indicate FALSE for this field; All CMEM v1.1, and later nodes, MUST indicate their level of support for schedules.
"queryParmsSupported"	This element indicates whether, or not, the source device supports the use of query string parameters on GET requests to the /PSIA/Metadata/stream resource (see ***Section 10.2.4*** following). The parameters, and their usage, are described in the 'stream' section.

A metadata/event source offers metadata to subscribers via the formats and session types advertised by the "MetaSessionSupport" schema.

The PSIA ordained session types for transferring metadata/event information are advertised via the following string tags. ***Please note that the naming construct of each tag is based on the relationship of the session initiator versus the data sender.*** *This irrespective of the source and consumer relationships. The first five session types are about how the publisher/source can send data to another node. The last 3 session types are for publishers that advertise that they can receive metadata from another node where the data is 'pushed' to them.*

String/Tag Value	Notes
"RESTSyncSessionOutTargetSend"	The session initiator sets-up an outbound session to the Target node, who is the data sender. HTTP REST connection. Synchronous = Session initiator must issue a GET to the Target to initiate the data stream. This session type is described in ***Section 9.2***.
'RESTSessionSrcOutRawTCP'	Source and Consumer support a data-specific (i.e. no control info), raw TCP connection for metadata transfer. Source establishes connection to Consumer using supplied 'netAddress'. (see Section 9.8.1)
'RESTSessionIntoSrcRawTCP'	The same connection type as above except the Source acts as a Server and the Consumer establishes the connectiopn into the Source (see Section 9.8.2). Same formatting restrictions apply as above.
'RESTSessionSrcOutRawUDP'	UDP version of the raw TCP connection. Source initiates an outbound raw UDP connection to Consumer. May be unicast or multicast.

RESTSyncSessionInTargetRecv	The session initiator sets-up an inbound session to the Target node, who is the data receiver. HTTP REST connection. Synchronous = Session initiator must issue a GET to the Target to initiate the data stream. This session type is described in ***Section 9.2***.

As noted, for each protocol tag identifier above that represents an HTTP/REST based session protocol, there is a corresponding "metaSessionFlowType" element that indicates the flow types supported for each protocol selection. Details regarding the flow modes are described in ***Section 9.4***. Please review this section for details.

11.2.3 /PSIA/Metadata/channels

This resource returns a schema instance that describes the all of the input channels that are metadata/event sources. The primary function of channel-based descriptions of metadata/event sources is to allow consumers to differentiate between specific input channels, devices, or *processes*, when necessary. Please note, that an 'input channel' is any literal, or virtual process, or set of processes and/or inputs, that generates metadata/event data. This includes hardware ports, logical processes, or any grouping of resources that enables better management and filtering of metadata/event information via their respective sources. This structure allows users to 'filter' A secondary benefit is that this resource allows administrators to determine which 'channels' are internal, external or are processes, and discover their respective characteristics.

URI	/PSIA/Metadata/channels		**Type**	Resource
Function				
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<MetaChannels>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes	The 'GET' request is issued to return a 'MetaChannels' schema instance that describes all of the active metadata/event input sources.			

Examples

```
<MetaChannels version= »1.1 » >
        <numOfChannels>2</numOfChannels>
        <metaChannelList>
                <metaChannelParms>
                        <channelID>1</channelType>
```

```
                        <channelType>internalSource</channelType>
                        <metaFormatList>
                                <metaFormat>gmch-psia</metaFormat>
                        </metaFormatList>
                        <channelDescription>Video Motion Detection</channelDescription>
                </metaChannelParms>
                <metaChannelParms>
                        <channelID>2</channelID>
                        <channelType>localSource</channelType>
                        <metaFormatList>
                                <metaFormat>gmch-psia</metaFormat>
                        </metaFormatList>
                        <channelDescription>IO/Contact port</channelDescription>
                </metaChannelParms>
        </metaChannelList>
</MetaChannels>
```

In the above example, a simple device indicates it has 2 channels of metadata/event inputs. The first channel is an internal Video Motion Detection process that generates 'VideoMotion' metadata. The second channel is a local I/O port that drives a contact. These channels can be correlated back to the 'MetadataList' supplied by the device.

```
<MetaChannels version= »1.1 » xmlns= »urn :psialliance.org »>
        <numOfChannels>4</numOfChannels>
        <metaChannelList>
                <metaChannelParms>
                        <channelID>1</channelType>
                        <channelType>internal</channelType>
                        <metaFormatList>
                                <metaFormat>gmch-psia</metaFormat>
                        </metaFormatList>
                        <channelDescription>Video Motion Detection</channelDescription>
                </metaChannelParms>
                <metaChannelParms>
                        <channelID>2</channelID>
                        <channelType>local</channelType>
                        <metaFormatList>
                                <metaFormat>gmch-psia</metaFormat>
                        </metaFormatList>
                        <channelDescription>IO/Contact port</channelDescription>
                </metaChannelParms>
                <metaChannelParms>
                        <channelID>3</channelID>
                        <channelType>local</channelType>
                        <metaFormatList>
                                <metaFormat>gmch-psia</metaFormat>
                        </metaFormatList>
                        <channelDescription>Audio events</channelDescription>
                </metaChannelParms>
                <metaChannelParms>
                        <channelID>4</channelID>
                        <channelType>local</channelType>
                        <metaFormatList>
                                <metaFormat>gmch-psia</metaFormat>
```

```
                        </metaFormatList>
                        <channelDescription>Video events</channelDescription>
                </metaChannelParms>
                <metaChannelParms>
                        <channelID>5</channelID>
                        <channelType>local</channelType>
                        <metaFormatList>
                                <metaFormat>gmch-psia</metaFormat>
                        </metaFormatList>
                        <channelDescription>System events</channelDescription>
                </metaChannelParms>
        </metaChannelList>
</MetaChannels>


In the above example, a hypothetical encoder device has 5 input 'channels' of metadata information. The
first channel ("1") is the Video Motion Detection process on the encoder. The second channel ("2") is an
attached I/O (dry) contact port. The third channel ("3") is a virtual input channel that addresses audio
signal-related events. The fourth channel ("4") is another virtual input channel that reports video
signal related events. The final channel ("5") is the internal system process used to report boot-up
occurrences and internal system errors.
```

The 'metaChannels' resource object reports the specific characteristics, or properties, associated with each metadata/event source(s). For endpoints, this information describes either A) the internal processes that generate metadata, or B) the attached physical ports that receive, and process, forms of metadata, or C) some combination of the aforementioned (i.e a 'virtualSource'). For proxy devices, the "MetaChannels" schema indicates where and what the metadata/event sources are. This enables improved administration of data networking via visibility into the who/what/where attributes of sources versus proxies. The "MetaChannels" schema is a list of the active channels supported by a device or system. The first element in "MetaChannels" is "numOfChannels" which indicates, in advance, the number channel descriptor list elements that follow in the schema instance. The next section of the schema consists of a sequenced list (i.e. "metaChannelList") of one, or more, channel parameter descriptors. The elements that comprise each channel parameter descriptor, "metaChannelParms", are described in the following table.

Element Name	Notes
"metaChannelParms":: "channelID"	The ASCII unsigned integer that is the 'handle' for the respective channel.
"metaChannelParms":: "channelType"	An enumerated string that identifies the type of input source that 'drives' a channel. Choices are: • "internalSource": and internal process generates the metadata. • "localSource": A physical port, of some type, is the input, thought it may require internal processing, for the metadata. An example would be Point-Of-Sale data that is introduced via device's serial port. • "remoteSource": This channel is a proxy for some remote source.

	• "activeBroadcast": This channel is an active multicast/broadcast channel that a consumer may 'tune' into. This tag represents an output channel unlike the other tags. It is used so that devices can describe active multicast channels.
"metaChannelParms":: "metaFormatList"	A list of the MIDS/URI categories and priorities that are related to the associated channel. This is the same XML 'type' that is used in the "metadataList" schema (***Section 10.2.1***)
"metaChannelParms":: "channelSrcGUID"	If the above 'channelType' = "remoteSource" then the Proxy device MUST list the GUID/UUID of that remote device.
"metaChannelParms":: "channelSrcINetAddress"	If the above 'channelType' = "remoteSource" the Proxy device MUST list the IP network address of that source device.
"metaChannelParms":: "multicastAddress"	If the 'channelType' = "activeBroadcast" the source/proxy MUST list the multicast address, and port number, that correlate to that broadcast channel/session.
"metaChannelParms":: "channelDescription"	A user-friendly descriptive string that helps describe the channel further.
"metaChannelParms":: "transactionAck"	Nodes that require that metadata/event occurrences be ACK'd at transaction level (i.e. either transactional ACKing or stateful ACKing), MUST indicate that requirement in this field.
"metaChannelParms":: "virtualSourceDescription"	This element/object indicates that a 'channel' is virtual/logical (i.e. it is comprised of a set of literal and/or logical process(es)). Please note that there may be more than one virtual source comprising a 'channel'.
"metaChannelParms":: "virtualSourceDescription":: "virtualSourceResourceID"	This element identifies the REST resource that is one of the component parts of the associated 'channel'. This is a required field.
"metaChannelParms":: "virtualSourceDescription":: "virtualSourceCategory"	All virtual source components MUST list the type of source that they are. Please reference the list of items in "vSrcCategory" XML type in "metaChannels.xsd".
"metaChannelParms":: "virtualSourceDescription":: "virtualSrcSubscripting"	All virtual source components that enable subscripting (see below) MUST indicate so. Otherwise, this field is left 'False'.
"metaChannelParms":: "virtualSourceDescription"::	For ALL virtual sources that have 'sub-channels' (i.e. sub elements) that can be addressed individually, or in

“virtualSrcSubrange”	sets, the source MUST provide the valid subscripting range so that subscribers/ consumers may know this valid range. For example a ‘partition’ virtual source that had 15 partitions would indicate a valid range of 0-14 or 1-15 as applicable.
“metaChannelParms”:: “virtualSourceDescription”:: “virtualSrcExplanation”	Virtual Source SHOULD list an explanation / description of what the virtual source represents.

The metadata/event channel information provided by each publisher is not necessary for basic information. The ‘metadataList’ and ‘sessionSupport’ resource objects, with their accompanying schemas, should provide enough information for identifying, accessing, and consuming metadata/event information. Channel information is primarily useful for 2 cases: A) the ability to ‘filter’ metadata/event information based on a source (i.e. channel) set, and/or B) the administration and management of inputs.

11.2.4 /PSIA/Metadata/stream

This resource object is the session setup object for the PSIA Metadata resource hierarchy. All session parameters are ‘setup’ with the Metadata ‘stream’ object for all HTTP/REST managed sessions. The operation, and interaction, with this resource object is simple: a consumer, after reading the ‘metadataList’ and ‘sessionSupport’ information (see prior), issues a GET or POST (for asynchronous or secondary connections) to the “/PSIA/Metadata/stream’ resource object, with an accompanying “MetaSessionParms” schema instance, to setup a metadata stream session of some specified flavor that is compliant with the source node’s “sessionSupport” attributes.

URI	/PSIA/Metadata/stream		**Type**	Resource
Function	This resource is the access point on a Metadata source for setting-up Metadata/Event sessions.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	***Conditional:*** See Section 10.2.4.2 descriptions for QSP details.	<MetaSessionParms>	<ResponseStatus; if successful, it has the session ID in the “id” field>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	***Conditional:*** See Section 10.2.4.2 descriptions for QSP details.	<MetaSessionParms>	<ResponseStatus; if successful it has the session ID in the “id” field>	

DELETE	None	None	Delete is only where a consumer wants to terminate the current synchronous session after setting parameters for non-synchronous session.
	This Metadata/stream resource is the access point for establishing Metadata/Event streams. They are setup in one of the following 2 manners: • 'GET' is used for retrieving information on the same session, synchronously; • 'POST' is used to setup (i.e. create) an asynchronous, or a separate, session for transferring Metadata/Event information. • A 'POST' may be used to setup a synchronous session if the Consumer cannot send a MetaSessionParms XML payload with a GET message, and the target does not support QSPs. In these cases, a POST is issued with the MetaSession-Parms, an 'id' is returned in the ReponseStatus, the Consumer issues a GET to the /PSIA/Metadata/stream/<id> resource/interface using the assigned 'id' value. See Section 9 for details and examples on the various session flows.		

Example(s)

```
<MetaSessionParms version="2.0">
       <MetaXportParms>
              <metaSessionID>0</metaSessionID>
              <metaFormat>gmch-psia</MetaFormat>
              <metaSessionProtocolType>RESTSyncSessionOutTargetSend</metaSessionProtocolType>
              <metaSessionFlowType>datastream</metaSessionFlowType>
       </MetaXportParms>
</MetaSessionParms>
```

In the above example, a consumer, initiating an HTTP/REST session to a target source desires to utilize that same session to retrieve a metadata/event stream in GMCH format. If the target/source node accepts the session parameters, it returns the same schema instance with an updated Session ID, and the consumer will then start receiving any active metadata/event information from the source.

```
<MetaSessionParms version="2.0">
       <MetaXportParms>
              <metaSessionID>0</metaSessionID>
              <metaFormat>gmch-psia</MetaFormat>
              <metaSessionProtocolType>RESTSyncSessionOutTargetSend</metaSessionProtocolType>
              <metaSessionFlowType>datastream</metaSessionFlowType>
              <metadataNameList>
                     <metadataIDString>/psialliance.org/VideoMotion</metadataIDString>
                     <metadataIDString>/psialliance.org/System</metadataIDString>
                     <metadataIDString>/psialliance.org/Config/update</metadataIDString>
              </metadataNameList>
       </MetaXportParms>
</MetaSessionParms>
```

The above example is based on the prior example. However, in this case the consumer only wants to see VideoMotion and System category events, plus specifically wanting notification of configuration updates that may occur ("/psialliance.org/Config/update").

```
<MetaSessionParms version="2.0">
```

```
        <MetaXportParms>
                <metaSessionID>0</metaSessionID>
                <metaFormat>xml-psia</MetaFormat>
                <metaSessionProtocolType>RESTAyncSessionBackSourceSend</metaSessionProtocolType>
                <metaSessionlogin>
                        <authMode>digest</authMode>
                        <userLogin>
                                <userName>gandalf512</userName>
                                <password>wizardsRule</password>
                        </userLogin>
                </metaSessionLogin>
                <metaSessionFlowType>datastream</metaSessionFlowType>
                <netAddress>
                        <targetIPAddress>206.14.5.70:3016</targetIPAddress>
                </netAddress>
                <metadataChannelList>
                        <metaChannel>1</metaChannel>
                        <metaChannel>2</metaChannel>
                </metadataChannelist>
                <metaSessionPersistence>dynamic</metaSessionPersistence>
        </MetaXportParms>
</MetaSessionParms>
```

In the above example the consumer wants the target/source to subsequently connect back to it at IP/Port address 206.14.5.70/3016, with the login user name and password of “gandalf512/wizardsRule”, and send metadata/event information to it in PSIA XML format for all metadata occurring on/from channels 1 and 2 (no specific metadata categories are provided). Also note that this asycnh-session is ‘dynamic’, i.e. it is demand driven (see Section 9.9).

NOTE: User login information (user name/password/credentials), when required, MUST never be passed across in an un-encrypted session. The only sessions that may require user login information are the asynchronous HTTP/REST sessions. Login information is invalid for all other session types.

```
<MetaSessionParms version=”2.0”>
        <MetaXportParms>
                <metaSessionID>0</metaSessionID>
                <metaFormat>gmch-psia</MetaFormat>
                <metaSessionProtocolType>RESTRTPStreamSrcOutUDP</metaSessionProtocolType>
                <netAddress>
                        <targetIPAddress>239.240.108.32:5004</targetIPAddress>
                </netAddress>
                <netcastMode>multicast</netcastMode>
                <metadataNameList>
                        <metadataIDString>/psialliance.org/System</metadataIDString>
                        <metadataIDString>/psialliance.org/Config</metadataIDString>
                </metadataNameList>
                <metadataChannelList>
                        <metaChannel>1</metaChannel>
                        <metaChannel>2</metaChannel>
                </metadataChannelist>
                <metaSessionPersistence>static</metaSessionPersistence>
        </MetaXportParms>
</MetaSessionParms>
```

In the above example the consumer wants the target/source to stream metadata/event information on an RTP/UDP multicast connection (i.e. IP address 239.240.108.32, port 5004). The information to be streamed is System and Configuration category metadata from channels 1 and 2 of the source device. Note the

```
concurrent use of metadata name, and channel, lists to filter information for the respective connection.
```

The "/PSIA/Metadata/stream' resource object is the stream 'access point' for initiating sessions for the transfer of metadata/event information in compliance with the attributes advertised by a source device, or system, via its "/PSIA/Metadata/sessionSupport" object (see Section 10.2.2 above). The 'stream' object receives the "MetaSessionParms" schema instance with the session parameters, validates the parameter values against the device's capabilities, and, if there is compatibility, initiates the data transfer using the session paramters. Since there are multiple session types, the initiation of data transfer is 'session type' dependent. The examples listed above outline how the parameters related to session initiation are used. Fundamentally, a consumer reads a node's 'MetaSessionSupport' definitions (via the "/PSIA/Metadata/sessionSupport' object), determines what the session and format capabilities are, and then initiates data transfer by 'GET'ing a session from the "/PSIA/Metadata/stream" resource object by also sending the compatible "MetaSessionParms" values. The elements in the "MetaSessionParms" schema are described in the table below.

Element Name	Notes
"metaXportParms"	Root element that contains the session and format parameters
"metaXportParms":: "metaSessionID"	Requesters always set this ID value to zero (which is equivalent to NULL). The successful setup of a session responds with and HTTP "200 OK" and the same schema instance with a valid (i.e. nonzero) session ID.
"metaXportParms":: "metaFormat:	Parameter that indicates the format of the metadata to be transferred. Choices are: "xml-psia": For those nodes that have XML defined metadata/event schemas. Note: legacy devices have defined gmch-psia. This should not be used for any new development.
"metaXportParms":: "metaSessionType:	This required parameter specifies the session protocol/transport type to be initiated for metadata transfer. These types are discussed in detail in Section 9 of this document. The tag strings used as values are listed below (and in the XSD above): • "RESTSyncSessionTargetSend": Required. See Section 9.2. • "RESTAsyncSessionBackSourceSend": Required. See Section 9.3. • "RESTRTPStreamSrcOutUDP": Optional except for RaCM devices where it is Required. See Section 9.5. • "RESTRTPStreamSrcOutTCP": Optional. See Section 9.5 • "RTSPRTPStreamSrcOut": Optional See Section 9.5. • "RTSPRTPStreamSrcOutInterleaved": Optional. Reserved for future use. • "RESTAsyncSessionBackForReceive": Optional. Reserved for

	future use. • "RESTRTPStreamInUDP": Optional. Reserved for future use. • "RESTRTPStreamInTCP": Optional. Reserved for future use • **"RESTSessionSrcOutRawTCP": Optional. See *Section 9.8*.** • **"RESTSessionSrcOutRawUDP": Optional. See *Section 9.8*.**
"metaXportParms":: "metaSessionFlowType"	For HTTP/REST based session protocols (see above), the consumer MUST specify the flow type IF there is more than one mode supported. All consumers SHOULD provide this parameter for HTTP/REST connection setup.
"metaXportParms":: "sessionLogin"	**Only** the asynchronous HTTP/REST sessions *may* have a need for user login/credential information in order to setup the asynchronous notification sessions. When required, user name/password information MUST only be sent on an HTTPS (i.e. encrypted) session; never should this information be sent in clear text! The fields within this 'type' are described next…
"metaXportParms":: "sessionLogin":: "authMode"	(see above conditions) When session login information is present, the session initiator must provided the HTTP authentication/security mode to be used for notification session establishment. Choices are: "none": no authentication needed. "basic": HTTP basic authentication (RFC 2616) "digest": HTTP digest based authentication (RFC 2616). "https-SSLv3": HTTPS using SSLv3 "https-TLSv1": HTTPS using TLS 1.0 (RFC 2246) "https-TLSv1.1": HTTPS/TLSv1.1 (RFC 4346) "https-TLSv1.2": HTTPS/TLSv1.2 (RFCs 5246/5746) "https-TLSv1.x": HTTPS/TLS v1.x best effort compatibility between nodes "any": Rely on HTTP challenge/handshake "other": Proprietary security model On the above selections, the session initiator MUST provide the most detailed information possible. If the target device cannot support the session authentication mode, it must fail the session parameter setup attempt. Otherwise, session authentication compatibility won't be known until the initiation of the first notification session.
"metaXportParms":: "metaSessionLogin"::	The user name and password information is specific to asynchronous HTTP/REST notification sessions. If the node

“userLogin”:: (“userName” & “password”)	receiving the ‘callback’ notification session requires a login, this information MUST be provided. Otherwise it is optional. Please note that sessions that utilize credentials (see following) may, or may not, need login information.
“metaXportParms”:: “metaSessionLogin”:: “credentialObject”:: (“credentialType”, “credentialDefinition”, ‘xs:any’)	Asynchronous HTTP/REST notification sessions *may* require credentialing logic. Especially for SSL/TLS sessions. This XML type allows clients/consumers to pass credential information to the devices that they expect to initiate ‘callback’ notification sessions.
“metaXportParms”:: “metsSessionRole”	(New CMEM v1.1) If a session initiator is setting up multiple Asynchronous notification sessions, then the Role of each target recipient, ‘primary’ versus ‘alternate’, MUST be supplied. When a trigger or metadata event occurs, connections will be attempted to all ‘primary’ nodes. If a connection to a primary node fails, then connections are attempted to the specified alternates. If an initiator lists only one Asynchronous notification session, the target node is assumed to be ‘primary’; it is invalid to have a single session where a node is ‘alternate’.
“metaXportParms”:: “targetHostName” , or… “targetIPAddress” , or… “targetIPv6Address”	(Changed for CMEM v1.1) In cases where the consumer is either A) requesting a ‘callback’ session, or B) a multicast session, this element is required to define either the host name of the target, or IPv4 address or the IPv6 address, and potentially, the port number (multicast) of the connection to be setup by the source.
“metaXportParms”:: “netcastMode”	This element denotes the mode of transporting RTP/UDP data. Since the default is “unicast” mode, this field only needs to be present to denote “multicast” sessions. Please see the XSD, and examples, for more details.
“metaXportParms”:: “transactionAck”	If a source requires that receivers must acknowledge (‘ACK’) each metadata/event instance, it MUST indicate this mode of operation by indicating ‘TRUE’ for this element. The default for all sources is ‘FALSE’.
“metaXportParms”:: “metadataNameList”	A list of metadata categories that the consumers desires to receive. Basically, it’s a traffic filter that can optionally be applied on the data stream. This list can be used with the following list, also.
“metaXportParms”:: “metadataChannelList”	A list of channels, via channel IDs, that the consumers desires to receive metadata from. Basically, this is a traffic filter that can optionally be applied on the data stream. This list can be used with the above list, also.
“metaXportParms”::	Boolean field that indicates to consumers/subscribers whether

“multicastCapable:	the metadata/event source supports UDP multicast transports, or not.
“metaXportParms”:: “scheduleCapable”	Required Boolean field for future use. All Version 1.0/1.1 nodes MUST indicate ‘FALSE’.
“metaXportParms”:: “queryParms”	Boolean field that indicates whether or not the source node accepts Query String Parameters (QSPs) as part of the URI of an HTTP/REST message regarding metadata/event streams, or not. The default is ‘FALSE’. Only the presence of this element, *with* a value of ‘TRUE’, indicates QSPs are supported.
“metaXportParms”:: “metaSessionPersistence”	(**New v2.0**). This parameter is expressly asynchronous session types (see ***Sections 9.3, 9.9***). The settings indicate where an established session is to be demand-driven (‘dynamic’), which means it is to be disconnected during significant durations of inactivity (and re-established when activity occurs), or if it is to be maintained as a stable session once connected (‘static’).

11.2.4.1 /PSIA/Metadata/stream/<*ID*>

URI	/PSIA/Metadata/stream/<id>		**Type**	Resource
Function	PSIA REST resource/object that holds the session parameters for active metadata/event sessions. Also, active sessions can be shutdown in an orderly fashion (recommended) using the DELETE method with the ID of an active session.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	If HTTP Status code = 200 <MetaSessionParms> Else <ResponseStatus w/error code>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	None	None	<ResponseStatus> DELETE is only used to notify the Source that the Consumer is shutting down the Session identified by the ID value.	
Notes	The ‘GET’ request issued to retrieve an instance of the ‘MetaSessionParms’ XML schema such that session parameters can be retrieved for an active session. The ‘DELETE’ is used to notify the Source that an imminent termination of the referenced session is occurring.			

All active sessions have a corresponding ID value that was assigned by the Source at the time of initiation. A Client/Consumer may retrieve the session parameters governing an active session by issuing a 'GET' to the /PSIA/Metadata/stream/<id> resource with a valid session ID. Additionally, all Consumers SHOULD issue 'DELETE's, for a specific session, to a Source prior to session teardown as part of an orderly session shutdown process. This process has been in all CMEM versions. However, it was originally included in the "/PSIA/Metadata/stream' Method description table which caused some confusion. This section clarifies this resource in addition to the "/PSIA/Metadata/stream" resource described above.

11.2.4.2 Query String Parameters (new v1.1 and later)

CMEM Version 1.1 (and later) compliant nodes have the ability to support Query String Parameters (QSPs) on HTTP/REST 'GET' or 'POST' messages sent to a node's '/PSIA/Metadata/stream' resource. The primary benefit of this feature is the ability to allow a 'shortcut' or 'shorthand' option for session setup where the following conditions are all true:

- A) The source has indicated it supports QSPs in its "MetaSessionSupport" XML parameters (see the /PSIA/Metadata/sessionSupport resource definition in *Section 10.2.2*); and…
- B) The consumer is setting up an HTTP managed session (i.e. non RTSP) ; and…
- C) The consumer has no more than **7** of the QSPs listed in the parameter table outlined below, and in Section 10.2.4.2.1 (history related QSPs).
- D) No "MetaSessionParms" XML document is supplied for session parameters; QSPs and XML documents are mutually exclusive. When QSPs are used, they alone provide the parameter set for a session.

In these cases, the consumer/client is allowed to supply the session parameters via QSPs. The allowable QSPs are listed below.

Table] HTTP/REST Stream QSPs

QSP Name	Requirement	QSP Description
"format"	Required	This QSP tag is used to identify which metadata format is to be used for transferring data. The choices are: o "xml-psia" See "MetaSessionParms" for details.
"channel"	Optional	This QSP tag indicates, which channel(s), if any, is/are to be the source(s). There may be multiple of these QSPs.
"category"	Optional	This QSP indicates which metadata categories the consumer desires to receive. The value is in MIDS format (see Section 7.2). E.g. "/psialliance.org/System/boot". Shorthand notation is allowed. E.g. "category=//VideoMotion" wild-cards the domain and states that all VideoMotion occurrences are interesting to the consumer: There may be multiple of these QSPs (up to the limit).
"class"	Optional	This QSP is shorthand notation of the above "category" QSP.

		Basically, it omits the domain field while still utilizing the MIDS format. Some examples: • "//System": Indicates the consumer wants all System metadata/events. • "//VideoAnalytics/Alerts": Indicates that the consumer wants all VideoAnalytics (Class) Alerts (Type). • "//Config/update: Indicates that the consumer wants to be notified of configuration update events. All configuration events would be "//Config". The above notation assumes that a source is either a pure PSIA node, or that other potential domains are either synonomous in their classifications, or there are no other matches whatsoever.
"output"	Optional	For PSIA metadata/event categories where the level of content sent can be specified by an output 'profile' level. This QSP tag designates the appropriate profile level. Values are: o "basic" o "full" PSIA Working Groups specify output profiles/levels, as needed, in their specs. Currently, the Video Analytics WG, uses this feature.
"ip"	Optional	This QSP is only valid for 'POST' messages that setup Asynchronous sessions. The "ip" value specifies the full IP address where the Consumer/Subscriber wants to receive data. Some valid examples are: "ip=181.14.70.12" : IPv4 address, assumed port 80 "ip=145.208.31.9:2106" : IPv4 address using port 2106

The above QSPs are the direct equivalent of the RTSP extension header fields described in *Section 9.6.2.1*. Please reference this area of this specification, for any additional details. The QSPs used for accessing historical metadata/event information are listed in the next section (10.2.4.2.1).
An example URI that uses QSPs in a compliant manner follows:

```
GET /PSIA/Metadata/stream?format=xml-
      psia&category=/psialliance.org/VideoAnalytics/Alert&output=full
      HTTP/1.1
```

Another version of the above URI, using the shorthand 'class=' notation (i.e. the domain is assumed/'wild-carded') for the URI is:

```
POST /PSIA/Metadata/stream?format=xml-
      psia&class=//VideoAnalytics/Alert&output=full&ip=190.112.17.23:8080
      HTTP/1.1
```

11.2.4.2.1 Stream Query String Parameters (QSPs) and History Transmission

In addition to the QSPs described in the prior section, the /PSIA/Metadata/stream resource supports QSPs for session initiation via the GET ***and POST*** methods. The allowed QSPs are listed in the table below.

QSP Name	*Parameters*	*Notes*
"send="	"history", "all"	The 'send' QSP is used to instruct an event/metadata server to either history data, or history data and live metadata information (i.e. continue the session after transmitting history info for transmission of subsequent information).
"time="	XML dateTime formatted timestamp	When accessing historical metadata/event information, consumers are *required* to specify the time from which the server/source is to start sending the historical metadata/event information, unless they desire to consume ALL of the history data.

The above QSPs are additional to those specified in Section 10.2.4.2, and may be used in conjuction with them, or with a MetaSessionParms XML body, to setup sessions where historical information is to be transferred. Parameters related to the contents, time range, and amount of history information are accessible via the '/PSIA/Metadata/history' resource (see Section 10.2.5). Below are some examples of how the parameters operate.

Example	*Explanation*
`GET /PSIA/Metadata/stream?send=history…`	The Consumer wants to get ALL of the historical metadata/event information, but does not want to continue the session for live information.
`POST /PSIA/Metadata/stream?send=all&time=2012-12-05T21:32:52…`	The Consumer wants the Source/Server to create an asynchronous session and send all historical event information from 9:32PM December 12th, 2012 (and more recent) then continue to send
`GET /PSIA/Metadata/stream?send=all…`	The Consumer wants the Source/Server to send ALL metadata/event information from the oldest historical information on to continuing the session for live events.

Please note that in the above example the XML document body (MetaSessionParms) containing the session parameters has been omitted for the sake of brevity. Also it is significant to note that ALL nodes implementing '/PSIA/Metadata/history' (see below) must support QSPs (see prior section).

11.2.4.3 /PSIA/Metadata/clientStream

URI	/PSIA/Metadata/clientStream		Type	Resource
Function	Clients can POST 1-n AreaControlEvents for scenarios where the REST Client is acting as a source of event information.			
Methods	Query String(s)	Inbound Data	Return Result	
POST	N/A	N/A	<ResponseStatus w/error code>	
Notes	Clients can POST 1-n AreaControlEvents.			

Example(s)

```
<AreaControlEventList>
  <AreaControlEvent>
…
  </AreaControlEvent>
  <AreaControlEvent>
…
  </AreaControlEvent>
</AreaControlEventList>
```

The clientStream resource is used by the Simple Reliable POST method of streaming as the destination URI for POSTing payloads.

11.2.5 /PSIA/Metadata/history (new for v2.0)

URI	/PSIA/Metadata/history		Type	Resource
Function	This resource reports the key statistics for a node's metadata/event history buffer or queue.			
Methods	Query String(s)	Inbound Data	Return Result	
GET	None	None	<MetadataHistory>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes				

Example(s)

```
<metadataHistory> version=”1.0”>
        <metaHistoryCount>24</metaHistoryCount>
        <oldestInfo>2012-12-10T08:28:14.09Z</oldestInfo>
        <newestInfo>2012-12-10T11:53:08.14Z</newestTime>
        <totalHistorySize>7238</totalHistorySize>
        <historyLog>false</historyLog>
</metadataHistory>
```

In the above example, the source node currently has 24 elements/records in its metadata/event history buffer. The oldest is from 8:28AM GMT December 10, 2012 and the newest event is from 11:53AM GMT Dec. 10th, 2012. The total data size of all 4 events is 7,238 bytes. This node does not have a history log.

```
<metadataHistory> version=”1.0”>
        <metaHistoryCount>8</metaHistoryCount>
        <oldestInfo>2012-12-12T13:28:14.09Z</oldestInfo>
        <newestInfo>2012-12-12T16:53:08.14Z</newestTime>
        <totalHistorySize>2271</totalHistorySize>
        <historyLog>true</historyLog>
        <historyLogCount>112</historyLogCount>
</metadataHistory>
```

In the above example, the source node currently has 8 elements/records in its metadata/event history buffer. The oldest is from 1:28PM GMT December 12th, 2012 and the newest event is from 4:53PM GMT Dec. 12th, 2012. The total data size of all 4 events is 2,271 bytes. This node does have a metadata history log and the log currently has 112 elements/records in it.

The /PSIA/Metadata/history resource provides the key statistical information on what is currently in a node’s metadata history buffer. Additionally, the information indicates if the node has a metadata history log, or not. And, if it does, it renders the number of items currently present in that log. It is significant to note that a ‘read’ of the history resource renders a ‘snapshot’ of the current saved metadata which is ephemeral with respect to time and potential event occurrences. Nodes that maintain a history log, in addition to the basic statistical information defined here, implement the ‘/PSIA/Metadata/history/log’ resource which is defined below.

MetadataHistory Parameters

Parameter/Element Name	*Req’d / Optional/ Dependent*	*Note*
metadataHistory::metaHistoryCount	R	Number of Events in the History buffer/queue.
metadataHistory::oldestInfo	R	Date/Time of oldest event information.
metadataHistory::newestInfo	R	Date/Time of newest event information
metadataHistory::totalHistorySize	R	Size, in bytes, of all Event history information
metadataHistory::historyLog	R	Flag indicating whether there is a history log, or not
metadataHistory::historyLogCount	D	If there is a log, the number of elements in the log

The event information listed above enables consumers to determine how much metadata/event information is available, and when that data occurred. This information can be retrieved via the '/PSIA/Metadata/stream' resource. The history log information, described above, indicates to consumers whether there is a log, or not, and if so, how many elements are in that log. The history log resource is described next.

11.2.5.1 /PSIA/Metadata/history/log

URI	/PSIA/Metadata/history/log		Type	Resource
Function	This resource contains a log of 'n' number of metadata/event occurrences. Each record/element has the metaID and the timestamp.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	"span=	None	<MetadataHistoryLog>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes				

Example

```
<metadataHistoryLog> version="1.0">
       <metaHistoryLogCount>3</metaHistoryLogCount>
       <metaHistoryRecord>
              <logID>42103</logID>
              <metaID>psialliance.org/System/fault/01a5cc06</metaID>
              <metaTime>2012-12-10T08:28:14.09Z</metaTime>
       </metaHistoryRecord>
       <metaHistoryRecord>
              <logID>42104</logID>
              <metaID>psialliance.org/System/boot</metaID>
              <metaTime>2012-12-10T08:28:21.30Z</metaTime>
       </metaHistoryRecord>
       <metaHistoryRecord>
              <logID>42105</logID>
              <metaID>psialliance.org/Network/connectionActive</metaID>
              <metaTime>2012-12-10T08:28:26.83Z</metaTime>
       </metaHistoryRecord>
       <metaHistoryRecord>
              <logID>42106</logID>
              <metaID>psialliance.org/Network/sessionActive</metaID>
              <metaTime>2012-12-10T08:30:09.01Z</metaTime>
       </metaHistoryRecord>
</metadataHistoryLog>

In the above example, the source node currently has 3 elements/records in its metadata/event history
blog. All of the events are from December 12th, 2012. The oldest occurred at 8:28.14AM GMT; this was a
system fault and the diagnostic code '01a5cc06' accompanied the fault (mfgr specific). The next event,
```

```
roughly 7 seconds later, was a system boot event (the node has rebooted from the fault). This was
followed 5.5 seconds later with a network connection active event. Roughly a minute and a half later a
management session was setup to the device. Please note that 'log IDs' are only required to be
monotonically increasing integers. The ones depicted above are based on a historical log count that
exceeds the current number of entries.
```

The history log resource object maintains a record of each metadata/event instance that has occurred. The duration and size capacity of each log is up to the implementer, but the log should strive to maintain a meaningful amount of information. A day's worth of activity is recommended, if possible. Minimally, a log should handle enough data to span a worst-case network outage. Please note that the size of the history log in '/PSIA/Metadata/history/log' and the size of the history buffer in '/PSIA/Metadata/history' are not linked. A log only ***has*** to hold the metaID and timestamp information for each event whereas the history buffer has to hold the complete information related to an event occurrence.

Since the size of a metadata history log can be large in some cases, all nodes that support history logs must render the current log element count in the "historyLogCount" element of the 'MetadataHistory' document supplied by the '/PSIA/Metadata/history' resource. Additionally, the '/PSIA/Metadata/history/log' resource supports the '***List Access Management***' methods described in ***Section 10.6***, of the ***PSIA Service Model v2.0*** specification. This method allows consumers to specify start/stop log IDs for piece-wise transmission of list data.

Metadata History Log Parameters

Parameter Name	*Required/ Optional/ Dependent*	*Notes*
metadataHistoryLog:: metaHistoryLogCount	**R**	Number of elements/records in the metadata history log.
metadataHistoryLog:: metaHistoryRecord	**D**	If the log count is greater than zero, then log elements/records MUST be present.
metadataHistoryLog:: metaHistoryRecord:: logID	**R**	Sequential, unsigned integer uniquely identifying each log record/element.
metadataHistoryLog:: metaHistoryRecord:: metaID	**R**	The metaID/MIDS of the metadata/event occurrence listed by an element/record.
metadataHistoryLog:: metaHistoryRecord:: metaTime	**R**	The timestamp of the metadata/event instance listed in the log element/record.
metadataHistoryLog:: metaHistoryRecord:: metaTime	**O**	Optional descriptive information associated with a metadata/event occurrence.

11.2.6 /PSIA/Metadata/broadcasts (optional resource)

This resource advertises the presence of any active multicast sessions that a source node may be broadcasting. Only nodes that have their "multicastCapable" value set to "true" in their "MetaSessionSupport" schema instance ("/PSIA/Metadata/sessionSupport"; ***Section 10.2.2***) are required to have this resource. Any systems/devices that do not advertise the ability to do multicast metadata streams MUST not have this resource object present. As with many other resources, this object's only function is to publish the attributes of active broadcasts that consumers may desire to connect to. Details are covered below.

URI	/PSIA/Metadata/broadcasts		Type	Resource
Function	This resource reports the active broadcast/multicast sessions, along with their session attributes, emanating from a source.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<MetaBroadcasts>	
PUT	N/A	N/A	<ResponseStatus>	
POST	N/A	N/A	<ResponseStatus>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes				

Example(s)

```
<MetaBroadcasts version="1.1">
       <numOfBroadcasts>2</numOfBroadcasts>
       <broadcastList>
              <broadcastSession>
                     <streamID>1001</streamID>
                     <multicastAddress>239.206.11.30:2112</multicastAddress>
                     <metadataNameList>
                            <metadataIDString>/PSIA/Config</metadataIDString>
                            <metadataIDString>/PSIA/System</metadataIDString>
                     /metadataNameList>
              </broadcastSession>
              <broadcastSession>
                     <streamID>1002</streamID>
                     <multicastAddress>239.206.11.30:2114</multicastAddress>
                     <metadataNameList>
                            <metadataIDString>/PSIA/VideoMotion</metadataIDSting>
                            <metadataIDString>/PSIA/Video</metadataIDSting>
                            <metadataIDString>/PSIA/Audio</metadataIDSting>
                     /metadataNameList>
              </broadcastSession>
       </broadcastList>
</MetaBroadcasts>

In the above example, a source node is advertising that it has 2 active broadcast, metadata sessions.
```

The first, has a ‘stream ID’ of “1001” and is broadcasting to IPv4 address “239.206.11.30” and UDP port number “2112”. This broadcast stream carries Configuration and System related metadata/events so it carries more ‘administrative’-type event data. The second channel has a ‘stream ID’ of “1002” and is outputting data in IPv4 address “239.206.11.30” via UDP port number “2114.” This broadcast stream carries 3 categories of metadata: A) Video Motion events, B) Video signal events, and Audio signal events.

```
<MetaBroadcasts version="1.0" xmlns="urn:psialliance-org">
	<numOfBroadcasts>1</numOfBroadcasts>
	<broadcastList>
		<broadcastSession>
			<streamID>75</streamID>
			<multicastAddress>FF38:0:8000:788:0FFFF:7000:4190</multicastAddress>
		</broadcastSession>
	</broadcastList>
</MetaBroadcasts>
```

In the above example, a source node is advertising that it has a single active broadcast metadata session. This broadcast stream, has a ‘stream ID’ of “75” and this stream is broadcasting to IPv6 address “FF38:0:8000:788:0FFFF:7000” and UDP port number “4190”. The lack of either a matadata name list, or channel list, indicates that the source is sending all of its metadata/event information out on this broadcast stream. Therefore, a consumer would have to read the source’s “/PSIA/Metadata/metadataList” resource object to know the metadata/event categories that are active for that source.

The “/PSIA/Metadata/broadcasts” resource object is a read-only, *optional* resource. For source’s that support multicast transmission capabilities (i.e. ‘multicastCapability = true’ in the “MetaSessionSupport” schema instance) this resource is required. If a node does not support multicast transmission of data, then this resource is not required and should be absent from the node’s resource hierarchy. Basically, the “MetaBroadcasts” schema parameters advertise the information necessary for consumers to connect to already active multicast sessions carrying metadata. The use of RTSP as a setup mechanism is not required since this resource carries the equivalent information of an SDP descriptor instance. The parameters in the “MetaBroadcasts” schema are described in the following table.

Element Name	Require-ment Level	Notes
“numOfBroadcasts”	Required	Parameter indicating the number of active multicast sessions. It also indicates the number of entries that following in the schema instance’s broadcast session descriptor list (see following).
“broadcastList”	Required	List container for the broadcast session descriptors that describe the attributes of active multicast sessions.
“broadcastSession”	Required	For each active multicast metadata session, this descriptor contains and describes the parameters with a specific session instance.
“broadcastSession”:: “streamID”	Required	A source-unique unsigned integer that is the ‘handle’ for this session. Currently, this field is a placeholder for future use when RTSP may be used to setup metadata sessions.

“broadcastSession”:: “multicastAddress”	Required	The IPv4 or IPv6 multicast address, and the target UDP port number, for the active multicast session. This information is required such that consumers take this info and ‘join’, then connect to, active multicast metadata sessions.
“broadcastSession”:: “metadataNameList”	Optional	The source MUST advertise the metadata/event categories of an active multicast session unless it is sending ALL of its advertised metadata on that broadcast. This optional list supplies the metadata categories, in URI string format (MIDS), active on a broadcast stream.
“broadcastSession”:: “channelList”	Optional	Sources that allow, or enable, multicasting by (input) channel selection MUST advertise the channels that are active on a broadcast session unless ALL of their channel metadata is being broadcast (i.e. there is not subset being streamed).

For source’s supporting multicast, there are 2 manners in which multicast sessions can be initiated: A) via session initiation per the session parameters outlined in the prior 2 sections of this document, or B) via configuration (i.e. a device is configured to auto-start multicast sessions as part of its system bring-up process). Either way, the information contained in the “MetaBroadcasts” schema instance is the catalogue of active sessions. Please note that sources that are multicast capable, but do not have any active broadcast sessions yet, advertise a “numOfBroadcasts” value of zero (0) until a session is created.

11.3 */PSIA/Metadata/Actions (optional service hierarchy)*

This service describes the ‘actions’ and ‘events’ a device or system offers in conjunction with metadata processing. The term ‘actions’ addresses the management of metadata and event processing. Management consists of the scheduling, triggering, and notification methods associated with ‘events’. This includes signaling to both PSIA and non-PSIA nodes. The term ‘events’ indicates the special processing that may be assigned to specific metadata categories when they occur. This ‘special processing’ is usually a form of notification or signaling that is automatically setup to occur during an event.

As a PSIA compliant Service, the ‘Actions’ resource has the requisite ‘index’ and ‘description’ resources that advertise the attributes of the ‘actions/events’ service. It is important that all devices and systems that employ this optional ‘Actions’ service hierarchy clearly advertise which resources they support. Please note that this service/resource hierarchy supersedes, and replaces, the “/Custom/Event” service/resource hierarchy defined in Section 7.15 of the IPMD v1.1 specification.

11.3.1 /PSIA/Metadata/Actions/index

The PSIA required 'index' resource is defined below for the "/PSIA/Metadata/Actions" service

URI	/PSIA/Metadata/Actions/index		Type	Resource
Function	PSIA Mandatory REST resource/object that enumerates the 1st level child resources for '/Metadata'.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<ResourceList>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes	The 'GET' request issued to retrieve an instance of the 'ResourceList' XML schema. See the Service Model specification, Section 7, 8, 10, for schema details.			

Example

```
<ResourceList version="1.0"
              xmlns:xlink="http://www.w3.org/1999/xlink"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns="urn:psialliance-org"
              xsi:schemaLocation="urn:psialliance-org
                                  http://www.psialliance.org/schemas/system/1.0/service.xsd">
  <!-- See PSIA Service Model specification, Section 10.1.2 -->
  <Resource xlink:href="/Metadata/Actions/index">
    <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification, Section
10.1.2), since index is a required resource within a Service -->
    <name>index</name>
    <type>resource</type>
  </Resource>
  <Resource xlink:href="/Metadata/Actions/description">
    <!-- EXEMPLARY: NOT required in actual response (see PSIA Service Model specification, Section
10.1.2), since description is a required resource within a Service -->
    <name>description</name>
    <type>resource</type>
  </Resource>
  <Resource xlink:href="/Metadata/Actions/triggers">
    <name>Event trigger settings</name>
    <type>resource</type>
  </Resource>
  <Resource xlink:href="/Metadata/Actions/schedule">
    <!-- PSIA optional resource within a Service -->
    <name>Event scheduling settings</name>
    <type>resource</type>
  </Resource>
  <Resource xlink:href="/Metadata/Actions/notification">
    <name>Event notification types and settings</name>
    <type>service</type>
    <!-- indexr would recursively return nested resources -->
  </Resource>
</ResourceList>
```

11.3.2 /PSIA/Metadata/Actions/description

The 'description' resource is as follows. Please note that it gives the base level actions, and relevant schemas associated with the "/PSIA/Metadata/Actions" service's first level resources.

URI	/PSIA/Metadata/Actions/description		Type	Resource
Function	PSIA REST resource/object that describes the functional behavior of the "Metadata/event" service resource (see PSIA Service Model Sections 7, 8, 10 for more details).			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<ResourceDescription>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes	The 'GET' request issued to retrieve an instance of the 'ResourceDescription' XML schema. See the Service Model specification, Section 7, 8, 10, for schema details.			

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<ResourceDescription version="1.0"
                     xmlns:xlink="http://www.w3.org/1999/xlink"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xmlns="urn:psialliance-org"
                     xsi:schemaLocation="urn:psialliance-org

http://www.psialliance.org/schemas/system/1.0/service.xsd">
  <name>/PSIA/Metadata/Actions</name>
  <type>service</type>
  <get>
    <queryStringParameters>none</queryStringParameters>
    <inboundXML>none</inboundXML>
    <function>Metadata Action(s) service</function>
    <returnResult>ResourceDescription</returnResult>
    <notes>none</notes>
  </get>
  <put></put>
  <post></post>
  <delete></delete>
  <name>/PSIA/Metadata/Actions/triggers</name>
  <type>resource</type>
  <get>
    <queryStringParameters>none</queryStringParameters>
    <inboundXML>none</inboundXML>
    <function>Metadata Action(s) trigger settings</function>
    <returnResult>EventTriggerList</returnResult>
    <notes>none</notes>
  </get>
  <put>EventTriggerList</put>
  <post></post>
```

```
  <delete></delete>
  <name>/PSIA/Metadata/Actions/schedule</name>
  <type>resource</type>
  <get>
    <queryStringParameters>none</queryStringParameters>
    <inboundXML>none</inboundXML>
    <function>Metadata Action(s) schedule settings</function>
    <returnResult>EventSchedule</returnResult>
    <notes>none</notes>
  </get>
  <put>EventSchedule</put>
  <post></post>
  <delete></delete>
  <name>Metadata/Actions/notification</name>
  <type>resource</type>
  <get>
    <queryStringParameters>none</queryStringParameters>
    <inboundXML>none</inboundXML>
    <function>Metadata event notification methods/settings</function>
    <returnResult>EventNotificationMethods</returnResult>
    <notes>none</notes>
  </get>
  <put>EventNotificationMethods</put>
  <post></post>
  <delete></delete>
</ResourceDescription>
```

10.3.3 /PSIA/Metadata/Actions/attributes

Each device, or system, supporting the Metadata 'Actions' service MUST also support an 'attributes' resource. The function of this resource is to advertise to prospective consumers what level of functional support is present on any given metadata/event source. The information below describes the operational behavior of the "/PSIA/Metadata/Actions/attributes" resource.

URI	/PSIA/Metadata/Actions/attributes		**Type**	Resource
Function	Metadata/Event 'Actions' service's functional attributes.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<ActionsAttributes>	
PUT	N/A	N/A	<ResponseStatus w/error code>	
POST	N/A	N/A	<ResponseStatus w/error code>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes	This resource defines the /PSIA/Metadata/Actions functional support provided by a source. Consumers interrogate this resource to determine what function is provided for event management and notification.			
Actions' Service Attributes XSD (filename="actionsAttributes.xsd")				

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:psialliance-org" xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns="urn:psialliance-org" version="1.0">

<xs:element name="ActionsAttributes">
       <xs:complexType>
              <xs:sequence>
                     <xs:element name="maxTriggers" minOccurs="1" maxOccurs="1"
                            type="xs:unsignedInt"/>
                     <xs:element name="maxSchedules" minOccurs="1" maxOccurs="1"
                            type="xs:unsignedInt"/>
                     <xs:element name="maxNotifications" minOccurs="1" maxOccurs="1"
                            type="xs:unsignedInt"/>
                     <xs:element name="notificationTypeList" minOccurs="1" maxOccurs="1"
                            type="NotificationTypeList"/>
                     <xs:element name="snapshotSupport" minOccurs="1" maxOccurs="1"
                            type="xs:boolean"/>
                     <xs:element name="videoClipSupport" minOccurs="1" maxOccurs="1"
                            type="xs:boolean"/>
              </xs:sequence>
       </xs:complexType>
</xs:element>

<xs:complexType name="NotificationTypeList">
       <xs:sequence>
              <xs:element name="notificationType" minOccurs="1" maxOccurs="unbounded"
                     type="NotificationMethod"/>
       </xs:sequence>
</xs:complexType>

<xs:simpleType name="NotificationMethod">
       <xs:restriction base="xs:string">
              <xs:enumeration value="Email"/>
              <xs:enumeration value="IO"/>
              <xs:enumeration value="RESTAsyncSessionBackSrcSend"/>
              <!-- The following are IPMD legacy/deprecated transports  -->
              <xs:enumeration value="HTTP"/>
              <xs:enumeration value="FTP"/>
              <!-- The following are known transports with config'n outside the scope of
PSIA                  CMEM -->
              <xs:enumeration value="IM"/>
              <xs:enumeration value="Syslog"/>
       </xs:restriction>
</xs:simpleType>
</xs:schema>
```

As noted by the above information, the ‘attributes’ is a read-only resource. Sources indicate the functional level of support they offer for event notification. Also indicated are the functional limits, or capacity, of some of the information used for event notification. The parameters of this resource’s schema are described individually, below.

Element Name	Required/ Optional/ Conditional	Description
"ActionsAttributes:: maxTriggers"	Required	This element indicates the maximum number of triggers allowed by a source. All sources MUST support a minimum of **4 (four)** triggers; 8 triggers, or greater, is recommended.
"ActionsAttributes:: maxSchedules"	Required	This element indicates the maximum number of schedules allowed by a source. All sources MUST support at least **1 (one)** schedule. Two schedules, or more, are highly recommended.
"ActionsAttributes:: maxNotifications"	Required	This element indicates the maximum number of notification method instances allowed by a source. All sources MUST support a minimum of **6 (six)** notification method instances. More items may need to be supported based on the notification method types that a source can activate (see next).
"ActionsAttributes:: notificationTypeList"	Required	This element is a list of the notification method types supported by a source. All sources MUST support at least **1 (one)** asynchronous ***session*** notification method. The notification method types are: • **"RESTAsyncSessionBackSrcSend"**: PSIA CMEM REST/HTTP sessions to remote notes. ***Sources MUST support this notification method.*** • **"Email"**: Email sessions to remote nodes. • **"IO"**: Generation of local I/O output signaling. • **"HTTP"**: Raw HTTP sessions to remote nodes. • **"FTP"**: FTP sessions to remote nodes. • **"IM"**: Instant Messaging (reserved for future use) • **"Syslog"**: System logging, via a Syslog session, of events (reserved for future use).
"ActionsAttributes:: snapshotSupport"	Required	This element indicates whether, or not, a Source has the ability to provide video 'snapshots' with an event notification. For those that are capable, JPEG must be supported, minimally. Optionally, an I-frame from the codec type advertised in "/PSIA/Streaming/channels/<id>" may be used. Snapshots are used by the PSIA-REST, Email, HTTP and FTP notification method types (see later). Incorporation of snapshots into the GMCH framework for REST Asynch sessions is also an option. MIME types, or file extensions

		(FTP), indicate the format of supplied snapshot video content.
"ActionsAttributes:: videoClipSupport"	Required	This element indicates if a Source has the ability to provide video clips with an event notification. For Sources that provide video clips, the 'MP4' format (ISO/IEC 14496-14) MUST be supported (minimally); this format includes objects such as attachment or files. As with snapshots, MIME types, or file extensions (FTP), indicate the format of video clip contents. Please note that MP4 files/objects contain internal headers and parameters that identify codec types, profiles and other properties.

CMEM compliant sources, that support the 'Actions' service, MUST meet the above requirements. Recommendations should be adhered to unless significant resource constraints prevent implementation.

11.3.3 /PSIA/Metadata/Actions/schedules

The 'schedules' resource allows event consumers to set particular time spans for when a device should perform asynchronous notifications (i.e. PSIA sessions, Email, raw HTTP or FTP sessions, I/O signaling, etc…) for specific categories of metadata/event occurrences. The "/PSIA/Metadata/Actions/schedules" resource is actually a list 'container' that holds all of the currently configured 'schedules' setup on a particular device. The operational characteristics of the resource are described below.

URI	/PSIA/Metadata/Actions/schedules		**Type**	Resource
Function	Event schedules.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<MetaEventScheduleList>	
PUT	None	<MetaEventScheduleList>	<ResponseStatus>	
POST	N/A	<MetaEventSchedule w/zero ID>	<ResponseStatus w/new ID>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes	Defines the schedule. The schedule is defined as a date-time range and a set of time blocks that define when the events are active. If <ScheduleTimeRange> is not present, the schedule is always valid.			

The schema defines a list for all of the configured event schedules setup on a device or system. Please note that the "/PSIA/Metadata/Actions/attributes" resource advertises the maximum number of configured event schedules a particular device or system allows. The detailed definition of each event schedule element is described in the next section.

11.3.4 /PSIA/Metadata/Actions/schedules/<*ID*>

Each schedule is comprised of 2 major sections. The first, which is optional, is the "ScheduleTimeRange" which indicates in which date/time span the schedule is considered valid. Please note that when a "ScheduleTimeRange" is specified, that schedule is considered 'inactive' outside (i.e. prior and after) the designated date/time range. If a "ScheduleTimeRange" is not supplied, a schedule is considered valid from the moment it is created into perpetuity.
The next 'required' section in a schedule, is the actual day-of-week calendar data that determines when a source is supposed to trigger event notifications. For each day of the week, an entity may supply a simple timespan in XML time format, or a 'time map'. A 'time map' is a 24-character string where each hour of the day is indicated as being 'active' via a '1', or inactive via a '0'. Operational details are listed below.

URI	/PSIA/Metadata/Actions/schedules/<ID>		**Type**	Resource
Function	Event schedules.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	None	None	<MetaEventSchedule>	
PUT	None	<MetaEventSchedule w/ID>	<ResponseStatus>	
POST	N/A	<MetaEventSchedule w/zero ID>	<ResponseStatus w/new ID>	
DELETE	N/A	<MetaEventSchedule>	<ResponseStatus>	
Notes	Defines the schedule. The schedule is defined as a date-time range and a set of time blocks that define when the events are active. If <ScheduleTimeRange> is not present, the schedule is always valid.			

A schedule entails all of the time span information that is to be employed for governing triggers. Schedules do not reference triggers; triggers reference schedules (see next section). Therefore, consumers setup schedules, then configure/create event triggers that reference the corresponding schedule instance (via an 'ID'). Each schedule is comprised of the following parameters.

Element Name	**Description**
"MetaEventSchedule:: scheduleID"	Source assigned, unique ID for each schedule instance.
"MetaEventSchedule:: scheduleTimeRange" .. "MetaEventSchedule:: scheduleTimeRange::	Optional time range/limit that will govern the active lifespan of a schedule. If this element is not present, the subsequent schedule parameters are infinite; i.e. constantly recurring without a date/time limit. If this element is present, the "beginDateTime" and "endDateTime" parameters specify when a particular schedule

beginDateTime/endDateTime	instance is 'active'. After the "endDateTime" the schedule instance remains, but it has no effect.
"MetaEventSchedule:: timeBlockList"	List of all of the 'active' periods comprising a schedule. An 'active' period specifies when triggers are 'active' for generating notification.
"MetaEventSchedule:: timeBlockList:: timeBlock:: dayOfWeek"	Indicator for which day of the week the corresponding time information applies to. Days are listed via number where 1=Sunday, 2=Monday, 3=Tuesday,…,7=Saturday.
"MetaEventSchedule:: timeBlockList:: timeBlock:: timeSpanList:: timeSpan"	Each designated day must be scheduled either via A) time spans, or B) via a time map (see following). A time span uses XML time notation to give the begin/end times for each active period of a day (see above) in hours/minutes/seconds. One, or more time spans, comprise a time span list.
"MetaEventSchedule:: timeBlockList:: timeBlock:: timeMapString"	A consumer can optionally use a 'time map' to specify the active periods within a given day of the week. A time map is a 24 character string where each hour of the day is represented by a one ('1') or a zero ('0'). A one indicates an 'active' hour in the day, whereas a zero indicates an inactive hour in the day.

11.3.4.1 XML Example: Schedule event detection and triggering

The command below, recurringly schedules event detection and triggering from 8:00 AM to 6:00 PM and 10:00 PM to 11:00 PM every Sunday, Tuesday, and Thursday. On Monday and Wednesday, event detection and triggering is scheduled from 7:00 AM to 5:00 PM.

```
POST /PSIA/Metadata/Actions/schedules HTTP/1.1
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<MetaEventSchedule version="1.1">
       <scheduleID>0</dcheduleID>
       <timeBlockList>
              <timeBlock>
                     <dayOfWeek>1</dayOfWeek>
                     <timeMapString>000000001111111111000010</timeMapString>
              </timeBlock>
              <timeBlock>
                     <dayOfWeek>2</dayOfWeek>
                     <timeSpanList>
                            <timeSpan>
                                 <beginTime>07:00:00</beginTime>
```

```
                                <endTime>17:00:00</endTime>
                        </timeSpan>
                </timeSpanList>
        </timeBlock>
        <timeBlock>
                <dayOfWeek>3</dayOfWeek>
                <timeMapString>000000001111111111000010</timeMapString>
        </timeBlock>
        <timeBlock>
                <dayOfWeek>4</dayOfWeek>
                <timeSpanList>
                        <timeSpam>
                                <beginTime>07:00:00</beginTime>
                                <endTime>17:00:00</endTime>
                        </timeSpan>
                </timeSpanList>
        </timeBlock>
        <timeBlock>
                <dayOfWeek>5</dayOfWeek>
                <timeSpanList>
                        <timeSpan>
                                <beginTime>08:00:00</beginTime>
                                <endTime>18:00:00</endTime>
                        </timeSpan>
                </timeSpanList>
        </timeBlock>
        <timeBlock>
                <dayOfWeek>6</dayOfWeek>
                <timeSpanList>
                        <timeSpan>
                                <beginTime>22:00:00</beginTime>
                                <endTime>23:00:00</endTime>
                        </timeSpan>
                </timeSpanList>
        </timeBlock>
    </timeBlockList>
</MetaEventSchedule>
```

11.3.5 /PSIA/Metadata/Actions/triggers

The /PSIA/Metadata/Actions/triggers resource enables the ability to manage the conditions that drive actions (i.e. 'triggers'). These parameter sets identify the conditions, and the specified behavior, that occur when the specified conditions are met. Please note that the 'triggers' resource reports the list of all the active triggers when read (i.e. 'GET'). A client has the ability to create an event trigger item via 'POST' to the resource with an 'EventTrigger' document. Updating one, or more, Event Triggers is accomplished via a PUT of an "EventTriggerList" object. The allowable operations, and the XSD for the event trigger list, are described below.

URI	/PSIA/Metadata/Actions/triggers		Type	Resource
Function	Access the list of event triggers.			
Methods	Query String(s)	Inbound Data	Return Result	
GET	None	None	<EventTriggerList>	
PUT	None	<EventTriggerList>	<ResponseStatus>	
POST	None	<EventTrigger w/zero ID>	<ResponseStatus w/new ID>	
DELETE	None	None	<ResponseStatus>	
Notes	Event triggering defines how the device reacts to particular metadata/event occurrences, such as video loss or motion detection or I/O port state changes, etc. See the schema definition in ***Section 10.3.5*** for details of each trigger condition.			

The above schema definition is basically a list of the configured event 'triggers' maintained by a device's Metadata/Actions service. Each trigger has its own set of parameters governing the conditions that comprise an 'event' and the set of 'notifications' that are to be engaged once a trigger occurs. The definition of an event trigger follows in the subsequent section of this document.

11.3.6 /PSIA/Metadata/Actions/triggers/<ID>

All created event triggers are 'contained' in the /PSIA/Metadata/Actions/triggers' resource as elements of a trigger list. Individual 'trigger' instances are directly manageable via their respective IDs. Each 'trigger' is a set of parameters identifying the event triggering conditions and the (potential) actions that may be ascribed to those triggering conditions. Only GET, PUT and DELETE operations are allowed in specific trigger instances. The methods and definitions associated with event triggers are listed below.

URI	/PSIA/Metadata/Actions/triggers/<*ID*>		Type	Resource
Function	Access a particular event trigger.			
Methods	Query String(s)	Inbound Data	Return Result	
GET	None	None	<EventTrigger>	
PUT	None	<EventTrigger w/ID>	<ResponseStatus>	
POST	N/A	<EventTrigger w/zero ID>	<Response Status w/new ID>	
DELETE	None	<EventTrigger w/ID>	<ResponseStatus>	
Notes	An event trigger determines how the device reacts when a particular event is detected. <inputIOPortID> is only required if <eventType> is of the category "/psialliance.org/IO…".			

The schema definition contains the all of the parameters associated with the conditions qualifying an 'event' and the selectable actions that may be assigned to an occurrence. The elements in this schema are described, in more detail, below.

Name	***Description***
"EventTrigger"	Multi-parameter element that defines the conditions that constitute an event, and a set of potential actions to take when a trigger occurs.
"EventTrigger::id"	Source assigned unique ID for the respective event trigger item.
Choice: "EventTrigger:: eventCategories" ***Or***… "EventTrigger:: inputIOPortID"	All event triggers MUST have either: • A metadata/event category list, in MIDS format (see type description below), that constitute the metadata 'event' occurrences that cause a 'trigger' (with or without channel IDs for each category); or… • In I/O input port that will drive the event trigger. • Users must select one of the above stimuli.
"EventTrigger:: eventNotificationList"	All event triggers MUST drive some form of notification. This element comprises a list of one, or more, notification methods, referenced by the ID(s) of the notification method(s), that are to be activated when the above event condition occurs. Event notifications are covered in ***Sections 10.3.8 and 10.3.9.***
"EventTrigger:: eventScheduleID"	If an event trigger is to be governed via a 'schedule' the ID of that schedule must be entered to correlate the 'active' time spans of that schedule to the trigger event. Please note that 'scheduled' triggers only occur during 'active' time periods of a schedule. See ***Section 10.3.5*** for more information on Schedules.
"EventTrigger:: intervalBetweenEvents"	This parameters specifies the minimum interval between event triggers. If one, ore more events, succeed an event before the minimum interval has expired, they are ignored.
	Common Types
"EventTrigger:: EventCategoryList:: EventCategorySpec:: eventMetaID" and, optionally…	**eventMetaID**: The domain/class/type/SrcID/… of the metadata/event occurrence that is to drive a trigger. Please note that most categories include a channel, track, region, zone, or other ID in the "SrcID" slot. For those CMEM categories that do NOT include channel/track/region/zone IDs, and, the consumer wants to restrict which port/channel events are active from, the following optional parameter may be used. **srcChannelID**: If a consumer/client needs to restrict, or

"EventTrigger:: "EventCategoryList:: EventCategorySpec:: srcChannelID"	qualify, what source is to be the supplier of the above metadata./event category data, then this field may be supplied.

Basically, each trigger item is comprised of a stimulus, the responses, and the qualifiers. A stimulus is comprised of either: A) one or more metadata/event categories, or B) an I/O port that may go active. When a stimulus occurs, the 'notification list' refers to the IDs of the respective notification mechanisms to be generated upon the activation of a trigger. The qualifiers are:

- **Schedules**: Any event trigger may optionally be governed by a schedule. Schedules are setup via the "/PSIA/Metadata/Actions/schedules" resource. Once a schedule has been setup, an event trigger can reference that schedule instance, via its ID, such that the schedule will govern when a trigger is considered 'active' (i.e. will generate notifications).
- **Interval Time**: The interval time determines the minimum time duration that must elapse between an event and another occurrence, of the same type, before any trigger is activated. If 2 events occur in less time than specified by the interval time, the second event is ignored.

The information contained in each trigger item comprises all of the conditions and notification mechanisms that affect the operation of event triggers. The only other item that may affect the operation of an event trigger is the "/PSIA/Metadata/Actions/schedule" resource.

11.3.7 /PSIA/Metadata/Actions/notifications

The purpose of creating event triggers is to generate 'notifications'. The "/PSIA/Metadata/Actions/notifications" resource is the list container for all the notification mechanisms (known as 'methods') created on a device or system. The list is comprised of "EventNotificationMethods" that provide the detailed parameters for each form of notification employed. The specifics of the individual notification methods is covered in more detail in the next section. The operational details of the "/PSIA/Metadata/Actions/notifications" resource are described below.

URI	/PSIA/Metadata/Actions/notifications		Type	Resource
Function	Configure notifications.			
Methods	Query String(s)	Inbound Data	Return Result	
GET	None	None	<EventNotificationList>	
PUT	None	<EventNotificationList>	<ResponseStatus>	
POST	N/A	<EventNotificationMethod w/zero ID>	<ResponseStatus w/new ID>	
DELETE	N/A	N/A	<ResponseStatus w/error code>	
Notes	The following notification types are supported:			

11.3.8 /PSIA/Metadata/Actions/notifications/<ID>

The "/PSIA/Actions/Metadata/Actions/notifications/<ID>" resources each described a single method of activating a notification mechanism. Currently, the notification types that may be selected are comprised of 4 network session types, and I/O output signaling. Each notification method describes one, and only one, notification mechanism. The event triggers (see ***Sections 10.3.6/7***) 'point' to the specific notification methods by referencing the ID value(s) of the particular notification method(s) they want activated when that trigger condition occurs. Operational details are described below.

URI	/PSIA/Metadata/Actions/notifications/<ID>		Type	Resource
Function	Configure specific notification methods.			
Methods	Query String(s)	Inbound Data	Return Result	
GET	None	None	<EventNotificationMethod>	
PUT	None	<EventNotificationMethod>	<ResponseStatus>	
POST	N/A	<EventNotificationMethod w/zero ID>	<ResponseStatus w/new ID>	
DELETE	N/A	<EventNotificationMethod>	<ResponseStatus>	
Notes	The following notification types are supported:			

The schema definition contains all of the parameters associated with the asynchronous session types, and I/O signaling methods, that may be employed for event notification. Each event notification method may only define one notification method. Therefore, for each ID value, there is a one-to-one correlation between that ID and a specific notification method. As mentioned before, event triggers bind themselves to one, or more, notification methods by referencing the notification methods' ID values. The parameters associated with the notification method XSD are described below.

Element Name	Description
"EventNotificationMethod:: notificationMethodID"	Source assigned, unique ID for each notification method instance.
	Users MUST Select 1 of the following 5 Options
"EventNotificationMethod:: "PSIASessionNotification"	This selection activates the use of a PSIA Asynchronous Notification Session to other nodes as the method of notification. These session types are described in ***Sections 9.3, 10.2.2, and 10.2.4*** of this document. The selector MUST provide the session parameters outlined in the "MetaSessionParms" schema described in ***Section 10.2.4. Support for this notification method is required.***
"EventNotificationMethod:: "EmailNotification"	This selection activates the use of a basic Email session, initiated by the source, to notify other nodes of event activities. The Email session parameters are discussed in more detail later in this section.
"EventNotificationMethod:: "IOSignaling"	For those devices equipped with 5V I/O, this selection activates the use of direct I/O output as a signaling method when event activities occur. To offer this capability, sources have to support the /PSIA/System/IO…' resources described in the ***IP Media Device spec v1.1, Section 7.5.***
"EventNotificationMethod:: "HTTPNotification"	This selection activates the use of a simple, raw HTTP session, initiated by the source, to notify other nodes of event activities. The HTTP session parameters are discussed in more detail later in this section. *This method is considered a legacy method and is provided for backwards compatibility.*
"EventNotificationMethod:: "FTPNotification"	This selection enables the use of an FTP session to perform notification to other nodes of event activities. *This method is considered a legacy method and is provided for backwards compatibility.*
"EventNotificationMethod:: "notificationRecurrence"	For the above methods, a user may be offered the ability to determine when, and how often, notification, or signaling, may occur. The choices are: • "beginning": Notification/signaling occurs at the start of an event. ***This is the default***. • "end": Notification/signaling occurs at the end of an event. • "recur": Notification/signaling is to occur in a recurring manner (see following) whenever a session error is encountered (for session notifications) or as normal behavior for signaling notifications.
"EventNotificationMethod:: "notificationInterval"	If a consumer selects "recur" (see above), then the intervening time interval between recurrences. The units are in seconds.
	Common Types

"NetAddressType:: hostname …(or) ipAddress …(or) ipv6Address" "NetAddressType:: portNumber"	All session types that require contacting remote nodes (Email, HTTP, FTP), need to have the remote node's network address specified in one of the following formats: • "hostname": a DNS name. • "ipAddress": IPv4 address. • "ipv6Address": IPv6 address. In some cases, a user may need to specify a particular port number for a session.
"EventObjectParms:: metadataURIembedded" "EventObjectParms:: metadataXMLembedded" "EventObjectParms:: videoURIenabled" "EventObjectParms:: videoSnapshotEnabled" "EventObjectParms:: videoClipEnabled" "EventObjectParms::	'metadataURIembedded: For Email, raw HTTP or FTP this parameter indicates that the email body or HTTP payload or FTP file contents should consist of the MIDS of the metadata/event occurrence. For Email or HTTP the MIME is 'text/plain' 'metadataXMLembedded': Indicates for Email, HTTP and FTP that the body, payload or file should contain the metadata/event information in XML document format. This choice prevents the use of the embedded MIDS (see above). 'videoURIenabled': Indicates that the body, payload or file should contain a URI referencing a video clip or snapshot. This only works with the MIDS choice, not with other selections. 'videoSnapshotEnabled': Indicates that an Email attachment, HTTP payload, or FTP file should contain a video snapshot in one of the designated formats (see XSD for details). 'videoClipEnabled': Instead of a snapshot, a source may offer a pre/post video clip attached to Email, in an HTTP payload, or as an FTP file. The supported types are defined in the XSD. 'snapshotFormat': If snapshots are supported, the source must supply them in a known format. *JPEG is the preferred format (see XSD).* 'videoClipFormat': If video clips are supported they must conform to one of the known formats (see XSD). *ISO 14496-14*

snapshotFormat"	*(MP4) is the preferred format.* For sources that support video clips for event information, they may also provide the ability to configure the pre/post event capture duration. This is in tenths of seconds units (i.e. '.1' second).
"EventObjectParms:: videoClipFormat"	
"EventObjectParms:: preCaptureDuration/ postCaptureDuration"	

11.3.8.1 REST Asynchronous Session Parameters

When 'PSIASessionNotification' is the selected notification method, then a "RESTAsyncSessionBackSourceSend" session is the session/transport type to be employed for notification. The parameters associated with this session type are described in ***Sections 10.2.2 (advertised session parameters) and 10.2.4 (setting of the session parameters; this is used in the Event Notification Method schema used to define the notification session parameters) .***

In addition to the PSIA CMEM session parameters identified above, devices and systems that indicate they have the capability (see ***Section 10.3.3***) to supply video snapshots, or video clips, with event information, *should* provide the ability for users to select the attachment of video objects (snapshots or clips) to the metadata/event information conveyed in a PSIA session. The only valid "EventObjectParms" (see Common Types above) for PSIA notification sessions are:

Video Option Selection	Description
'videoSnapshotEnabled'	Indicates that the consumer wants snapshots to accompany XML event information. XML events will use a content type of "multipart/mixed".
'videoSnapshotFormat'	If video snapshot attributes are enabled, the format of the video content should be specified (unless the device/system only supplies one format).
'videoClipEnabled'	Indicates that the consumer wants video clips to accompany XML event information. XML events will use a content type of "multipart/mixed".
'videoClipFormat'	If video clip attributes are enabled, the format of the video content should be specified (unless the device/system only supplies the default format of MP4).

'pre/postCaptureDuration'	Optional support to configure duration (pre/post) of attached video clips.

Please note that the choices of video snapshot versus video clips are mutually exclusive for devices and systems that offer both options.

11.3.8.2 Email Notification Method Parameters

When Email is the selected notification method ("EmailNotification"), the parameters below define the session and content attributes for Email notification.

Element Name	Description
"EmailNotification:: receiverEmailAddress"	Email name/account of the target recipient.
"EmailNotification:: senderEmailAddress"	Email name/account to be used by the sender for this notification instance.
"EmailNotification:: subjectLine"	Subject line content to be listed for each Email notification.
"EmailNotification:: bodySetting"	Sources may offer the following settings for an Email body and/or attachment: ***Body Settings --*** • **metadataURIembedded:** Email body consists of the MIDS of the event. May be used with video URI also (see below). • **metadataXMLembedded**: Email body consists of an XML document containing the event information. If this option is chosen, no other body settings may be selected. • **videoURIenabled**: Source provides event video via a URI reference in the Email body. The format must be selected using the snapshot or video clip format options. ***Attachment settings --*** • **videoSnapshotEnabled**: Source provides a video snapshot as an attachment to the Email. Format MUST be one of the supported definitions (JPEG is preferred). Or,…. • **videoClipEnabled**: Source provides a video clip as an attachment to the Email. Video clip MUST be one of the supported formats. Sources may also provide pre/post capture duration parameters (see EventObjectParms in prior table).
"EmailNotification::	Consumer must select one of the following options:

mailAuthenticationMode”	• “**none**”: No authentication is to be performed for the Email session. • “**SMTP**”: SMTP authentication is to be used for the session. • “**POP/SMTP**”: POP/SMTP authentication is to be used for the Email session.
“EmailNotification:: emailAccountName”	If the remote, target account requires a login, then the account/user name needs to be provided.
“EmailNotification:: emailPassword”	If the remote, target account requires a login, then the password needs to be provided if one is required..
“EmailNotification:: networkAddress”	If required the network address (IP address or host name) of the target Email recipient.
“EmailNotification:: popServerHostName / popServerIPAddress / popServerIPv6Address”	If POP/SMTP is the chosen Email transport/ authentication mode (see above), then the address, via IP address or name, of the POP server needs to be provided.

11.3.8.3 HTTP Notification Method Parameters

When a raw/simple HTTP session is selected as the notification method (“HTTPNotification”) for event occurrences, the following parameters govern the session mechanics.

Element Name	Description
“HttpNotification:: sessionType”	User MUST select either “**http**” or “**https**” as the session type.
“HttpNotification:: networkAddress”	User MUST provide either the IPv4/IPv6 address, or host name, of the target HTTP recipient.
“HttpNotification:: bodySetting”	User MUST select from a choice of parameters that determine the structure of the HTTP POST signal. Options are: ***Body Settings --*** • **metadataURIembedded**: HTTP body/payload consists of the MIDS of the event. May be used with video URI also (see below). • **metadataXMLembedded**: HTTP body/ payload consists of an XML document containing the event information. If this option is chosen, no other body settings may be selected. • **videoURIenabled**: Source provides event video via a URI reference in the HTTP body/payload. If chosen, the format must be selected using the

	snapshot or video clip format options below. ***Video settings --*** • **videoSnapshotEnabled**: Source provides a video snapshot as the body of an HTTP message using the appropriate MIME type (e.g. 'image/jpeg'). Format MUST be one of the supported definitions (JPEG is preferred). Or,…. • **videoClipEnabled**: Source provides a video clip as the body of an HTTP message using the appropriate MIME type. Video clip MUST be one of the supported formats. Sources may also provided pre/post capture duration parameters (see EventObjectParms in preceding table).
"HttpNotification:: httpUserName / (and) httpPassword"	Sources are required to offer support for HTTP login accounts as an option. If the target HTTP recipient requires a login, the user MUST enter the name/password information.
"HttpNotification:: httpAuthenticationMethod"	Users MUST indicate if an HTTP notification sessions employs HTTP authentication. Choices are: '**none**': No HTTP authentication is needed. '**basic**': RFC 2616 Basic authentication is employed for the HTTP notification session. '**MD5diges**t': RFC 2616 Digest-based authentication is employed for the HTTP notification session.

11.3.8.4 **FTP Notification Method Parameters**

The use of FTP sessions to provide Event notification are defined by the parameters listed below.

Element Name	Description
"FtpNotification:: networkAddress"	User MUST provide either the IPv4/IPv6 address, or host name, of the target FTP server.
"FtpNotification:: ftpUserName"	User name of the FTP login required at the FTP server.
"FtpNotification:: ftpPassword"	Password of the FTP login required at the FTP server (may be empty; i.e. "").
"FtpNotification:: passiveModeEnabled	User MUST indicate if PASSIVE (true) or ACTIVE (false) FTP session mode is to be employed.
"FtpNotification:: ftpUploadPath"	Path of where the Source is to deposit the event information.
"FtpNotification:: ftpBaseFileName"	File name base to use when depositing event information. The data/time is appended to the end

	of this name to guarantee uniqueness. E.g. a base file name of "my-event-info" would render the following for event information occurring on June 21st, 2010 at 10:31:26.44 AM: "my-event-infoJune-21-2010-10.31.26.44.txt". Please note that the file extension will depend upon the file contents selected to be used (see next).
"FtpNotification:: ftpFileContents"	User MUST select from a choice of parameters that determine the contents of the FTP file sent. Options are: ***Body Settings --*** • **metadataURIembedded**: FTP file consists of the MIDS of the event. May be used with video URI also (see below). • **metadataXMLembedded**: FTP file consists of an XML document containing the event information. If this option is chosen, no other body settings may be selected. • **videoURIenabled**: Source provides event video via a URI reference in the FTP file contents. If chosen, the format must be selected using the snapshot or video clip format options below. ***Video settings --*** • **videoSnapshotEnabled**: Source provides a video snapshot as the contents of an FTP file using the appropriate file extension type (e.g. '*.jpg'). Format MUST be one of the supported definitions (JPEG is preferred). Or,…. • **videoClipEnabled**: Source provides a video clip as the contents body of an FTP file using the appropriate file extension type (i.e. '*.mp4' or '*.avi', etc.). Video clip MUST be one of the supported formats. Sources may also provided pre/post capture duration parameters (see EventObjectParms in preceding table). Unlike HTTP, FTP may offer both text/XML files AND video content at the same time. For Sources that offer this level of function, 2 files would be deposited per event occurrence. The file extensions would differentiate the contents whereas the file names would be the same (e.g. "myEvent-June-21-2010-10.31.26.44.xml" and "myEvent-June-21-2010-10.31.26.44.mp4" would indicate that an event deposited an XML event record *and* an MP4

	formatted video clip).

11.3.8.5 I/O Signaling Notification Method Parameters

I/O Signaling is a unique form of notification. For this mode of notification to be supported, Sources must support the PSIA "/System/IO/outputs" resource structures listed in the IP Media Device Specification v1.1 in ***Sections 7.5.5 – 7.5.9***. For automated signaling, based on events, the following parameters define this form of notification.

Element Name	Description
"IOsignal:: outputIOPortID"	The ID of the output I/O channel ("/PSIA/ System/IO/<id>) to be activated when an event occurs.

10.3.10 How Does This Trigger/Schedule/Notify Stuff Work?

The above subsections of Section 10.3 prescribe the data definitions, and basic mechanics, of how CMEM compliant metadata/event Sources provide varied asynchronous notification services. However, due to the overall size, and related complexity, it is usually difficult to understand without an overview of the basics. This section provides a high-level overview to cover the basic setup and correlation between the trigger, schedule and notification resources.

The diagram below provides a basic, reference example of some configured resources in the 'Metadata/Actions' service. The diagram also depicts to correlation and linkages between specific resource instances.

11.3.8.6 **Metadata/Actions Resource Relationship Diagram**

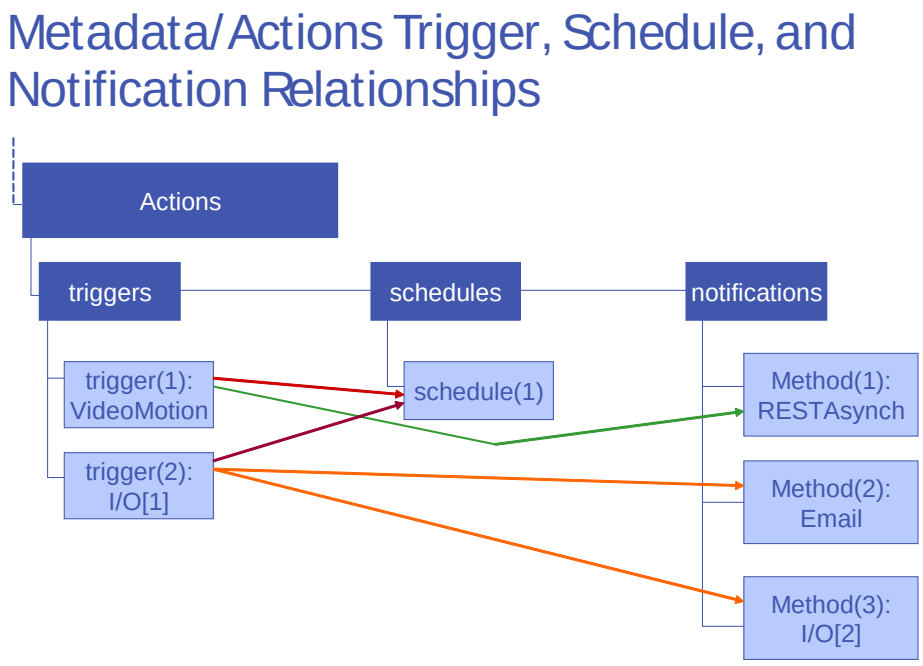


Figure 11-1 Metadata/Actions Resource Relationship Diagram

In the above example, 2 event triggers, 1 schedule, and 3 notification methods have been configured on a Source. The parenthesis in each resource block indicates the 'ID' of that respective resource. Each of the triggers is described below:

- "/PSIA/Metadata/Actions/triggers/1": This trigger ('trigger(1)') is setup to be active when "/psialliance.org/VideoMotion" events occur. Since no channel is specified in the MIDS (e.g. "/psialliance.org/VideoMotion//1"), all video motion detection occurrences will generate event triggers. 'Trigger(1)' references 'schedule(1)' as being the governing schedule for determining when 'trigger(1)' is active. When 'schedule(1)' indicates that 'trigger(1)' is active, and a video motion event occurs, 'trigger(1) is configured to generate a notification method. In this case, 'notification(1)' is the referenced notification method which is setup to generate a PSIA REST Asynchronous notification session (see ***Sections 9.3 and 10.2.2***). This is the only notification mechanism linked to 'trigger(1)'.
- "/PSIA/Metadata/Actions/triggers/2": This trigger ('trigger(2)') is setup as an I/O event trigger. It references I/O channel #1 ("/PSIA/System/IO/inputs/1") as the event stimulus. This trigger is also governed by the 'schedule(1)' resource for determining when it is considered active. *(Please note that many triggers can share any of the schedule and notification resources; there are no restrictions to the linkages between triggers and schedule/notification resources)* When I/O

input channel #1 goes active, and the schedule indicates that 'trigger(2)' is active, then 2 notification methods are employed: A) Notification method #2 ("/PSIA/Metadata/Actions/notifications/2") is setup as an Email notification; and B) notification method #3 ("/PSIA/Metadata/Actions/notifications/3") is setup as output I/O trigger for I/O output port #2 ("/PSIA/System/IO/outputs/2").

11.3.8.7 Example Setup of Triggers, Scheduling and Notifications

Using the reference example listed in the prior section (see above), a multi-step example is provided for reference. In this step-by-step example, 1 schedule, 2 event triggers, and 3 notification methods are configured conforming to the diagram in **Section 10.3.10.1**, above. Each phase is individually described in the subsections below. Please note that the examples provided are references, and though they employ many options, additional combinatorics are possible.

Step 1: Schedule Setup

Since 'triggers' reference both schedules (when employed), and notification methods, these items must be setup prior to configuring an event trigger. This scenario uses a single schedule to govern the active time spans for when event triggers are to occur. Step 1 consists of the creation of a schedule. The operation is as follows:

```
POST /PSIA/Metadata/Actions/schedules HTTP/1.1
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<MetaEventSchedule version="1.0" xmlns="urn:psialliance-org">
        <scheduleID>0</scheduleID>
        <timeBlockList>
                <timeBlock>
                        <dayOfWeek>1</dayOfWeek>
`                       <timeMapString>111111111111111111111111</timeMapString>`
                </timeBlock>
                <timeBlock>
                        <dayOfWeek>2</dayOfWeek>
                        <timeSpanList>
                                <timeSpan>
                                        <beginTime>18:00:00</beginTime>
                                        <endTime>07:00:00</endTime>
                                </timeSpan>
                        </timeSpanList>
                </timeBlock>
                <timeBlock>
                        <dayOfWeek>3</dayOfWeek>
                        <timeSpanList>
                                <timeSpan>
                                        <beginTime>18:00:00</beginTime>
                                        <endTime>07:00:00</endTime>
                                </timeSpan>
                        </timeSpanList>
                </timeBlock>
                <timeBlock>
                        <dayOfWeek>4</dayOfWeek>
                        <timeSpanList>
                                <timeSpan>
                                        <beginTime>18:00:00</beginTime>
                                        <endTime>07:00:00</endTime>
```

```
                                </timeSpan>
                        </timeSpanList>
                </timeBlock>
                <timeBlock>
                        <dayOfWeek>5</dayOfWeek>
                        <timeSpanList>
                                <timeSpan>
                                        <beginTime>18:00:00</beginTime>
                                        <endTime>07:00:00</endTime>
                                </timeSpan>
                        </timeSpanList>
                </timeBlock>
                <timeBlock>
                        <dayOfWeek>6</dayOfWeek>
                        <timeSpanList>
                                <timeSpan>
                                        <beginTime>18:00:00</beginTime>
                                        <endTime>07:00:00</endTime>
                                </timeSpan>
                        </timeSpanList>
                </timeBlock>
                <timeBlock>
                        <dayOfWeek>7</dayOfWeek>
`                       <timeMapString>111111111111111111111111</timeMapString>`
                </timeBlock>
        </timeBlockList>
</MetaEventSchedule>
```

The above schedule is unbounded. Once created, it never expires unless it is explicitly deleted. The schedule itself has active time spans from 6 PM until 7 AM for weekdays, and is active 24 hours a day on Saturdays and Sundays. When POST'ed, this schedule has no ID number, as referenced above. Once created, the source assigns this schedule instance an ID of '1' ("/PSIA/Metadata/Actions/schedules/1").

Step 2: Event Notification Method #1 Creation

The next step consists of the creation of a notification method to be employed by an event trigger. The operation is as follows:

```
POST /PSIA/Metadata/Actions/notifications HTTP/1.1
Content-type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<EventNotificationMethod version="1.1"">
    <notificationMethodID>0</notificationMethodID>
       <PSIASessionNotification>
           <PSIANotificationParms>
               <MetaSessionParms>
                   <MetaXportParms>
                       <metaSessionID>0</metaSessionID>
                       <metaFormat>gmch-psia</metaFormat>
                       <metaSessionProtocolType>RESTAyncSessionBackSourceSend
                               </metaSessionProtocolType>
                       <metaSessionlogin>
                           <authMode>digest</authMode>
                           <userLogin>
                               <userName>gandalf512</userName>
                               <password>wizardsRule</password>
                           </userLogin>
                       </metaSessionLogin>
                        <metaSessionFlowType>datastream</metaSessionFlowType>
                        <netAddress>
                           <targetIPAddress>206.14.5.70:3016</targetIPAddress>
```

```
                        </netAddress>
                    </MetaXportParms>
                </MetaSessionParms>
                <PSIASessionVideoSettings>
                        <videoSnapshotEnabled>true</videoSnapshotEnabled>
                        <videoSnapshotFormat>JPEG</videoSnapshotFormat>
                </PSIASessionVideoSettings>
            </PSIANotificationParms>
        </PSIASessionNotification>
    <notificationRecurrence>end</notificationRecurrence>
</EventNotificationMethod>
```

In the above step, a notification method is created for using a PSIA REST-based asynchronous session to IP address/port number "206.14.5.70:3016". The session format uses GMCH encapsulation and the configuration has setup JPEG snapshots to accompany the event information. Each session requires HTTP digest-based authentication along with a login (see above). Please note that once this notification method is created it will have an ID of '1' ("/PSIA/Metadata/Actions/notifications/1").

Step 3: Event Trigger Creation

In this step, an event trigger is created to define the conditions that will drive notification. The operation is listed below:

```
POST /PSIA/Metadata/Actions/triggers HTTP/1.1
Content-type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<EventTrigger version="1.1" xmlns="urn:psialliance-org">
        <triggerID>0</triggerID>
        <eventCategories>
                <eventCategory>
                        <eventMetaID>/psialliance.org/VideoMotion</eventMetaID>
                </eventCategory>
        <eventCategories>
        <eventNotificationList>
                <eventTriggerNotificationMethod>1</eventTriggerNotificationMethod>
        </eventNotificationList>
        <eventScheduleID>1</eventScheduleID>
        <intervalBetweenEvents>1</intervalBetweenEvents>
</EventTrigger>
```

The above event trigger parameters create a Video Motion trigger with a minimum trigger interval, between events, of 1 second. This event trigger instance is governed by 'schedule[1]' ("/PSIA/Metadata/Actions/schedule/1") and stimulate event notification method #1 ("/PSIA/Metadata/Actions/notifications/1") when the schedule is in an 'active' phase.

Step 4: New Notification Method #2

As part of the creation of a new event trigger, the user creates a new notification method using the following operation.

```
POST /PSIA/Metadata/Actions/notifications HTTP/1.1
Content-type: application/xml; charset="UTF-8"
```

```
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<EventNotificationMethod version="1.0" xmlns="urn:psialliance-org">
    <notificationMethodID>0</notificationMethodID>
        <EmailNotification>
                <receiverEmailAddress>alert@flimflam.com</receiverEmailAddress>
                <senderEmailAddress>cameraAtBackDoor@flimflam.com</receiverEmailAddress>
                <subjectLine>After hours I/O Alert!</subjectLine>
                <bodySetting>
                        <metadataXMLembedded>true</metadataXMLembedded>
                        <videoSnapshotEnabled>true</videoSnapshotEnabled>
                        <videoSnapshotFormat>JPEG</videoSnapshotFormat>
                </bodySetting>
                <mailAuthenticationMode>SMTP</mailAuthenticationMode>
                <emailAccountName>alertlogin</emailAccountName>
                <emailPassword>2soon2tell</emailPassword>
                <networkAddress>
                        <ipAddress>147.206.19.5</ipAddress>
                ,/networkAddress>
         </EmailNotification>
    <notificationRecurrence>beginning</notificationRecurrence>
</EventNotificationMethod>
```

In the above notification parameters, an Email session is defined. The target Email address is alert@flimflam.com. The sender's Email pseudonym is cameraAtBackDoor@flimflam.com. Each Email will have a subject line of "After hours I/O Alert!". The user has set the Email body to be XML event information. Additionally, JPEG snapshot attachments have been setup. SMTP protocol and login information has been provided for accessing the Email target at IP address "147.206.19.5". Each Email notification is to occur at the beginning of an event. Once created, this notification instance has an ID of '2' ("/PSIA/Metadata/Actions/notifications/2").

Step 5: Another Notification Method, #3, is Created

Another notification method is created as part of configuration. The operation is listed below.

```
POST /PSIA/Metadata/Actions/notifications HTTP/1.1
Content-type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<EventNotificationMethod version="1.1">
    <notificationMethodID>0</notificationMethodID>
        <IOSignaling>
                <outputIOPortID>2</outputIOPortID>
        </IOSignaling>
        <notificationRecurrence>recur</notificationRecurrence>
        <notificationInterval>1</notificationInterval>
</EventNotificationMethod>
```

The above notification method is simple. I/O output port #2, is driven active in recurring 1 second intervals. Please note that the device or system has to support a "/PSIA/System/IO/outputs/2" resource (see ***IPMD v1.1. Section 7.5***) for a notification method to 'drive' it. Once created, this notification method is assigned an ID of '3' ("/PSIA/Metadata/ Actions/notifications/3"). In this reference scenario, I/O output #2 is attached to an alarm.

Step 6: Event Trigger #2 is Created

As the last step, a new event trigger is created that utilizes both notification methods #2 and #3.

```
POST /PSIA/Metadata/Actions/triggers HTTP/1.1
Content-type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<EventTrigger version="1.1">
        <triggerID>0</triggerID>
        <inputIOPortID>1</inputIOPortID>
        <eventNotificationList>
                <eventTriggerNotificationMethod>2</eventTriggerNotificationMethod>
                <eventTriggerNotificationMethod>3</eventTriggerNotificationMethod>
        </eventNotificationList>
        <eventScheduleID>1</eventScheduleID>
        <intervalBetweenEvents>1</intervalBetweenEvents>
</EventTrigger>
```

This event trigger has I/O port #1 specified as its stimulus. This trigger is also governed by "/PSIA/Metadata/Actions/schedule/1" (just like the first event trigger). When an I/O event occurs on input port #1, in and 'active' time span, the trigger will cause an Email to be sent with a JPEG snapshot, and I/O output port # 2 will be driven to cause an alarm.

12.0How to Use CMEM (Metadata Services)

The 'Metadata' resources listed herein comprise the functional areas for accessing, apprising, and configuring metadata and event information. The overarching term for these capabilities and protocols is termed 'CMEM' which is the acronym for the 'Common Metadata/Event Model.' Since all PSIA sources are to use CMEM for the management of metadata and event information, it is important to ensure that an overall understanding of how it is intended to work is not lost in the data and protocol details. Below, a brief overview is provided regarding the basic steps and mechanisms for detecting and using CMEM functionality.

12.1 *Detecting Metadata Services/Resources*

All CMEM sources, like every other PSIA device, or source, MUST advertise themselves via mDNS/DNS-SD. Once a node is discovered, a consumer, or management entity (hereafter referred to as a 'client'), needs to interrogate the service and resource structure on a node. This is done via the use of the 'index' resource(s). All CMEM compliant nodes have the "/PSIA/Metadata/…" service, and subordinate resources, present which indicate that the node supports CMEM functionality.

12.2 *Detecting Metadata Functional Support*

Once the "/PSIA/Metadata" service has been detected, clients must read the 'description', 'metadataList', 'sessionSupport', and (optionally) the 'channels' resources to determine the basic functional capabilities supported by a CMEM node. Optionally, a node may also have the 'broadcasts' resource that indicates the active multicast sessions that are available for reception. Additionally, nodes are strongly encouraged to support the "/PSIA/Metadata/Actions" service. If the 'Actions' service is present, it will show up under the 'index' resource of the 'Metadata' service. 'Actions' is interrogated similarly to the 'Metadata' service. Its 'index' resource ("/PSIA/Metadata/Actions/index") indicates which resources are present.

12.3 *Detecting CMEM v1.1 versus v1.0 Functionality*

The easiest, quickest manner for detecting CMEM v1.1 functionality versus CMEM v1.0, is to read the "/PSIA/Metadata/sessionSupport" resource and interrogate the schema version. All CMEM v1.1. compliant nodes MUST have a "version=1.1" level in their 'MetaSessionSupport' document instance. Please note that this does not obviate the need to interrogate the other resources listed in the above sections.

12.4 *CMEM v1.1 Implementation Requirements*

The PSIA Common Metadata and Event Model (CMEM) v1.1 specification obsoletes, and deprecates, the prior CMEM v1.0 specification. ALL PSIA implementers should implement, or migrate to, CMEM v1.1 instead of version 1.0. This requirement is due to the following factors:

- The CMEM v1.1 specification has quickly followed the CMEM v1.0 spec.
- CMEM v1.1 has greater functionality and is aligned with the PSIA Video Analytics v1.0 specification.
- Migration to the latest standard aids interoperability.
- The functional delta between CMEM v1.1 and v1.0 is not large. In fact, the 'required' functional levels are almost identical.
- CMEM v1.1 has improved session management capabilities.

In addition to the above, the largest addition to CMEM v1.0, that is in CMEM v1.1, is the 'Actions' service and its resources. The goal of this addition was to standardize, and replace, the "/PSIA/Custom/Events" resources listed in the IPMD v1.0/v1.1 specifications such that all nodes could have common interfaces for scheduling and asynchronous notifications. CMEM v1.0 nodes would not have these capabilities.

13.0External Document References

[RFC 1305]	"Internet Time Synchronization: the Network Time Protocol", D. L. Mills, March 1992 URL:http://www.ietf.org/rfc/rfc1305.txt
[RFC 1945]	"Hypertext Transfer Protocol -- HTTP/1.0", T. Berners-Lee et al, May 1996 URL:http://www.ietf.org/rfc/rfc1945.txt
[RFC 2326]	"Real Time Streaming Protocol (RTSP)", H. Schulzrinne et al, April 1998 URL:http://www.ietf.org/rfc/rfc2326.txt
[RFC 2616]	"Hypertext Transfer Protocol -- HTTP/1.1", R. Fielding et al, June 1999 URL:http://www.ietf.org/rfc/rfc2616.txt
[RFC 2617]	"HTTP Authentication: Basic and Digest Authentication", J. Franks, et al, June 1999 URL:http://www.ietf.org/rfc/rfc2617.txt
[RFC 3339]	"Date and Time on the Internet: Timestamps", G. Klyne, et al, July 2002 URL:http://www.ietf.org/rfc/rfc3339.txt
[RFC 3550]	"RTP: A Transport Protocol for Real-Time Applications", H. Schulzrinne et al., July 2003 URL:http://www.ietf.org/rfc/rfc3550.txt
[RFC 4566]	"SDP: Session Description Protocol", M. Handley, et al, July 2006 URL:www.ietf.org/**rfc**/**rfc4566**.txt
[RFC 4571]	"Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport", J. Lazzaro, July 2006 URL:http://www.ietf.org/rfc/rfc4571.txt
[RFC 5285]	"A General Mechanism for RTP Header Extensions", D. Singer et al, July 2008 URL:http://www.ietf.org/rfc/rfc5285.txt
PSIA Service Model specification	www.psialliance.org/documents/PSIA-Service-Model_version_1_0.pdf, March 17, 2009
PSIA IP Media Device specification	www.psialliance.org/documents/PSIA-IPMD-v1r7.pdf, March 17, 2009

14.0Appendix B: CRC32 Source Code

C Source file :

```
/*---------------------------------------------------------------------*
  Copyright 2008, 2009
  Roger Richter

        This CRC32 source code is provided AS-IS with no expressed
        or implied warranties, guarantees, protections whatsoever. It
        solely provided as reference code, which may be used by 3rd
        parties at their own risk. Users of this code (i.e. those
        who compile and build executables with this code included
        irrespective of the form used) are granted a non-exclusive
        nontransferable worldwide distribution license in binary form
        as linked into an executable module whether that is a library,
        executable file, or other form of machine executable code.

        NOTES:
                - Code assumes a 32-bit lil-endian machine type!

 *---------------------------------------------------------------------*/

#include <crc32fw.h>

static const UINT32 crc32Table[] =
{
        0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f, 0xe963a535,
        0x9e6495a3,
        0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988, 0x09b64c2b, 0x7eb17cbd, 0xe7b82d07,
        0x90bf1d91,
        0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551,
        0x83d385c7,
        0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9, 0xfa0f3d63,
        0x8d080df5,
        0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172, 0x3c03e4d1, 0x4b04d447, 0xd20d85fd,
        0xa50ab56b,
        0x35b5a8fa, 0x42b2986c, 0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf,
        0xabd13d59,
        0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423, 0xcfba9599,
        0xb8bda50f,
        0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924, 0x2f6f7c87, 0x58684c11, 0xc1611dab,
        0xb6662d3d,
        0x76dc4190, 0x01db7106, 0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5,
        0xe8b8d433,
        0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d, 0x91646c97,
        0xe6635c01,
        0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e, 0x6c0695ed, 0x1b01a57b, 0x8208f4c1,
        0xf50fc457,
        0x65b0d9c6, 0x12b7e950, 0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3,
        0xfbd44c65,
        0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7, 0xa4d1c46d,
        0xd3d6f4fb,
        0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0, 0x44042d73, 0x33031de5, 0xaa0a4c5f,
        0xdd0d7cc9,
        0x5005713c, 0x270241aa, 0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409,
        0xce61e49f,
        0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81, 0xb7bd5c3b,
        0xc0ba6cad,
        0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a, 0xead54739, 0x9dd277af, 0x04db2615,
        0x73dc1683,
        0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27,
        0x7d079eb1,
        0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb, 0x196c3671,
        0x6e6b06e7,
```

```
        0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc, 0xf9b9df6f, 0x8ebeeff9, 0x17b7be43,
        0x60b08ed5,
        0xd6d6a3e8, 0xa1d1937e, 0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd,
        0x48b2364b,
        0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55, 0x316e8eef,
        0x4669be79,
        0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236, 0xcc0c7795, 0xbb0b4703, 0x220216b9,
        0x5505262f,
        0xc5ba3bbe, 0xb2bd0b28, 0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b,
        0x5bdeae1d,
        0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f, 0x72076785,
        0x05005713,
        0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38, 0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7,
        0x0bdbdf21,
        0x86d3d2d4, 0xf1d4e242, 0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1,
        0x18b74777,
        0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69, 0x616bffd3,
        0x166ccf45,
        0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2, 0xa7672661, 0xd06016f7, 0x4969474d,
        0x3e6e77db,
        0xaed16a4a, 0xd9d65adc, 0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f,
        0x30b5ffe9,
        0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693, 0x54de5729,
        0x23d967bf,
        0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94, 0xb40bbe37, 0xc30c8ea1, 0x5a05df1b,
        0x2d02ef8d
};

/*---------------------------------------------------------------------*
 * UINT32 crc32( crcParm, bufferPtr, bufLen );
 *
 * This function generates a CRC32 result for the buffer passed into
 * this routine. It can be used in chain mode by successively calling
 * it. Therefore, it assumes the CRC value passed into it has already
 * been setup/initialized (0xffffffff) by the caller prior to invocation.
 *---------------------------------------------------------------------*/

UINT32 crc32( UINT32 crcParm, UINT8 *pBuffer, UINT32 bufLen )
{
        while ( bufLen-- )
                crcParm = (crcParm >> 8) ^ crc32Table[((crcParm ^ *pBuffer++) & 0x000000ff)];
#if (__LIL_ENDIAN__)
        UNENDIAN( crcParm );
#endif
        return( (crcParm ^ CRC32MASKVAL) );
}
```

H file source (for the above):

```
/*---------------------------------------------------------------------*
 * Copyright 2008, 2009
 * Roger Richter

        This CRC32 source code is provided AS-IS with no expressed
        or implied warranties, guarantees, protections whatsoever. It
        solely provided as reference code, which may be used by 3rd
        parties at their own risk. Users of this code (i.e. those
        who compile and build executables with this code included
        irrespective of the form used) are granted a non-exclusive
        nontransferable worldwide distribution license in binary form
        as linked into an executable module whether that is a library,
        executable file, or other form of machine executable code.
 *
 * CRC32.H: Primary include file for CRC32.C source code
 *
 *---------------------------------------------------------------------*/

#if !defined(__CRC32FW__)
```

```
#define __CRC32FW__ 1


#if defined(__cplusplus )
       extern "C" {
#endif

#if defined(_MSC_VER)

       #define __LIL_ENDIAN__ 1
       #define __X86__ 1
       #define __ASM86__ 1
#elif
       /* In the non-MS Visual C/Studio case, you have to set the following
        * manually or insert your own compiler level detection.
        * BTE: I make a default assumption of little-endian x86 environment.
        */
       #define __LIL_ENDIAN__ 1
       #define __X86__ 1
       #define __ASM86__ 0

#endif

#define __REVENDIAN__ 0

/* If we're in a MS VC/x86 environment, use the special instructions, if reversing endian-ness is
needed */
#if (__REVENDIAN__)

#if (__ASM86__)
    #define UNENDIAN(val32) __asm    mov eax, val32 __asm   bswap eax __asm  mov  val32, eax
#else
       #define UNENDIAN(val32) \
                      val32 = ( ((val32 >> 24) & 0x000000ff) | ((val32 >> 8) & 0x0000ff00) | \
                             ((val32 << 8) & 0x00ff0000) | ((val32 << 24) & 0xff000000) )
#endif

#else

#define UNENDIAN(val32)

#endif

#if !defined(UIN64)
       #define UINT64 unsigned long long
#endif
#if !defined(UINT32)
       #define UINT32 unsigned int
#endif
#if !defined(UINT8)
       #define UINT8 unsigned char
#endif

#define CRC32POLYNOMIAL 0xedb88320
#define CRC32MASKVAL 0xffffffff
#define INITCRC(crc) crc = CRC32MASKVAL

/* Prototypes --------------------------------*/

UINT32 crc32( UINT32 crc, UINT8 *pBuf, UINT32 bufLen );

#if defined(__cplusplus)
}
#endif

#endif
```

15.0 Appendix D: Reserved SAH Tag Values

- “SYST”, 0x54535A53 (little endian): System State defined SAH (PSIA):
  - State Values are (Int32 for binary mode, XML integer):
    - 0 = Closed/Shutdown
    - 1 = Initializing/Opening/Booting
    - 2 = Active/Open/OK
    - 3 = Shutting-down/Closing/Resetting
    - 4 = Active/Open – Impaired/Limited
    - 5 = Stopped/Idle (normal)
    - -1 = Error/System Fault
- “ALST”, 0x54534C43 (little endian): Alarm State defined SAH (PSIA/Obix)
- “ACTL”, 0x4C544341 (little endian): Area Control defined SAH (PSIA)

16.0 Appendix 1: “VideoMotion” Metadata Class Dictionary

Binary Tag (CRC32) = 0x7194CE37

The following ‘Type’ fields/tags have been reserved within the ‘VideoMotion’ PSIA Metadata Class. Not all devices that support VideoMotion metadata/events have to support all of the Types below. These tags are reserved as currently known event definitions. The PSIA IP Media Device Working Group is the responsible party for publishing the required support.

- “**motionStart**” : Motion has started in an active ROI.
- “**motionStop**”: Motion has ceased in an active ROI.
- “**motion**” : Motion has occurred in an active ROI.
- “**motionROIActive**”: A ROI has become active.
- “**motionROIInactive**”: A ROI has gone inactive (for schedule based scenarios).
- “**motionUp**”: Upwards motion has been detected in a scene in an active ROI.
- “**motionDown**”: Downwards motion has been detected in an ROI.
- “**motionLeft**”: Left-wards motion has been detected in an ROI.
- “**motionRight**”: Right-wards motion has been detected in an ROI.
- “**continuousMotion**”: Run-on/continuous motion has been detected in an ROI.

An example of one of the above is: “`/psialliance.org/VideoMotion/motionStart`”.

17.0Appendix 2: “Video” Metadata Class Dictionary

Binary Tag (CRC32) = 0xBD06F528

The following ‘Type’ fields/tags have been reserved within the ‘Video’ PSIA Metadata Class. Not all devices that support ‘Video’ metadata/events have to support all of the Types below. These tags are reserved as currently known event definitions. The PSIA IP Media Device Group is the responsible party for publishing the required support.

- “**signalActive**” : A video signal on an input channel (hardware port; ‘DVR’ related) has gone active..
- “**signalInactive**”: A video signal on an input channel (hardware; ‘DVR’ related) has gone inactive (no signal).
- “**streamInactive**”: An input IP-based video stream has gone inactive (i.e. is ‘lost’; NVR related).
- “**streamActive**”: An input IP-based video stream has become active (NVR related).
- “**signalError**”: A video signal error, usually transient, has occurred on an input channel.
- “**allDark**” : An active scene has gone all dark (usually a normal light level to all dark)..
- “**allBright**”: An active scene has gone completely bright (usually from an overall dark/dim level).

An example of one of the above is: “`/psialliance.org/Video/signalError`”.

18.0 Appendix 3: “Config” Metadata Class Dictionary

Binary Tag (CRC32) = 0xD3262A4A

The following ‘Type’ fields/tags have been reserved within the ‘Config’ PSIA Metadata Class. This category of metadata indicates changes in the state of the configuration of a device or system.

- “**update**” : A configuration update/edit has occurred. The event should contain, at the end of the MIDS (past the Transaction ID field), the name of the schema updated if only one was modified.
- “**unauthorizedUpdate**”: A rejected, unauthorized configuration attempt occurred.

An example of one of the above is:
“/psialliance.org/Config/update/2//StreamingChannel”.

19.0 Appendix 4: “IO” Metadata Class Dictionary

Binary Tag (CRC32) = 0x8601BAB2

The following ‘Type’ fields/tags have been reserved within the ‘IO’ PSIA Metadata Class. This category of metadata indicates changes in the state of the configuration of a device or system. The owning PSIA working group for this metadata class is the IP Media Device group.

- “**active**” : An I/O port has gone active, from an inactive state.
- “**inactive”**:: an I/O port is in an inactive state; usually as a transition from an active state..
- “**IOError**”: An I/O error has occurred in the operation of the device/port.

An example of one of the above is: “`/psialliance.org/IO/active`”.

20.0Appendix 5: “Audio” Metadata Class Dictionary

Binary Tag (CRC32) = 0xD9BC1991

The following ‘Type’ fields/tags have been reserved within the ‘Audio’ PSIA Metadata Class. This category of metadata indicates changes in the state of the audio signal on a particular channel, or port. The owning PSIA working group for this metadata class is the IP Media Device group.

- “**signalActive**” : The audio signal for a specific channel/port has gone active from an inactive state..
- “**signalInactive”**: The audio signal has gone inactive from an active state for a particular channel/port.
- “**signalError**”: An audio signal error has occurred.
- “**levelHigh**”: The audio signal has gone from a quiet/low level to a high noise/sound level (usually set via a configurable threshold).
- “**levelLow**”: The audio signal has gone from a high or media noise level to a quiet or silent level (the level is usually set via a configurable threshold).

An example of one of the above is: “`/psialliance.org/Audio/levelHigh`”.

21.0 Appendix 6: “PointOfSale” Metadata Class Dictionary

Binary Tag (CRC32) = 0x9212C808

The following ‘Type’ fields/tags have been reserved within the ‘PointOfSale’ PSIA Metadata Class. This category of metadata indicates that a Retail-related Point-of-Sale transaction has occurred, of some form. Please note that not all ‘types’ are required to be supported. Devices should report the transactions they support with as much detail as possible. The owning PSIA working group for this metadata class is the Recording and Content Management (RaCM) group.

- “**sale**” : A normal/general transaction has occurred.
- “**void**”: A ‘void’ transaction has occurred.
- “**saleCash**”: A cash-based sale transaction has occurred.
- “**saleCredit**”: A credit-based sale transaction has occurred.
- “**saleCheck**”: A check-based sale transaction has occurred.
- “**saleDebit**”: A debit-card sale transaction has occurred.
- “**saleAmountTrigger**”: A sale that has exceeded a preset amount threshold has occurred.
- “**refund**”: A refund transaction has occurred.

An example of one of the above is: “`/psialliance.org/PointOfSale/void`”.

22.0 Appendix 7: "System" Metadata Class Dictionary

Binary Tag (CRC32) = 0xCEE114BD

The following 'Type' fields/tags have been reserved within the 'System' PSIA Metadata Class. This category of metadata indicates changes in the state of a device, or system, and any hardware or base system changes that may occur. The owning PSIA working group for this metadata class is the System group.

- "**boot**" : A device or system has booted-up (whether expected or not).
- "**fault"**: An unrecoverable system error has occurred.
- "**shutdown**": A device or system is in the process of shutting-down (i.e. a graceful reboot). This occurrence should also convey whether the occurrence was based on external, or internal, stimulus.
- "**versionUpdate**": A device or system has completed a successful version update. This message should be issued prior to 'shutdown' to let other nodes know that the update is about to commence. Therefore this event indicates the successful transfer and validation of version update binaries, not the actual post-update reboot.
- "**offline**": The device/node has been directed, either manually or programmatically, to go 'off-line'; i.e. the unit is active but the primary application is inactive. The 'system' management interface to the unit is active throughout this period.
- "**online**": The device/node, and its primary application, is active. This event usually occurs after the "offline' state (see above).

Environmental related items are below:

- "**tempTrigger**": A device or system has exceeded an internal temperature threshold.
- "**tamperAlarm**": An incident has occurred which has triggered a tamper alert/alarm.
- "**voltageError**": The device/node has encountered a voltage level error. For more detail, one of the following voltage definitions should be used.
- "**voltageLow**": A voltage underrun ("brown power") has occurred.
- "**voltageHigh**": A voltage overload has occurred.
- "**fanFailure**": A system/device/chassis fan has failed.
- "**fanSpeedError**": A system/device/chassis fan is not maintaining adequate air flow.
- "**batteryLow**": The system/device's battery is encountering a low power threshold.
- "**batteryFailed**": The system/device's batter has failed and needs to be replaced.
- "**shockAlarm**": A system/device has encountered a shock (i.e. g-force; accelerometer) threshold.
- "**powerSupplyFailed**": An internal power supply has failed. The unit is still operating (dual power supplies, etc.).
- "**upsActive**": A system/device is operating on UPS power (general power has failed).
- "**upsFailure**": A UPS power backup failure has occurred.
- "**powerNormal**": Normal power has been recovered.

An example of one of the above is: "`/psialliance.org/System/shutdown`".

23.0 Appendix 8: “Storage” Metadata Class Dictionary

Binary Tag (CRC32) = 0x9BC722A8

The following ‘Type’ fields/tags have been reserved within the ‘Storage’ PSIA Metadata Class. This category of metadata indicates changes in the state or operation of storage media. The owning PSIA working group for this metadata class is the RaCM group.

- “**mediaFailure**” : An unrecoverable storage media error has occurred; usually a disk failure .
- “**readError**”: This tag specifically identifies a storage media read error. Usually this indicates a recoverable or intermittent type error. Otherwise, “mediaFailure” should be indicated.
- “**writeError**”: This tag specifically identifies a storage media write error. Usually this indicates a recoverable or intermittent type error. Otherwise, “mediaFailure” should be indicated.
- “**newMedia**”: This tag indicates that a new storage media device, of some sort, has appeared on a device or system. This type is mostly oriented for the notification of the attachment of dynamic media types such as USB drives, CD-ROMs, DVDs, and external storage.
- “**mediaGone**”: This tag indicates a prior instance of storage media has been detached. This is useful in cases where a device or system knows storage media hardware is being removed (i.e. USB, CD/DVD, etc.).
- “**capacityThreshold**”: The storage has met, and probably exceeded, a set storage capacity threshold.
- “**mediaFull**”: The storage media (disk, volume, CD/DVD, etc.) is full. No more capacity is available.
- “**mediaUnformatted**”: The storage media (disk, volume, CD/DVD, etc.) is not formatted and unable to be written to.
- “**tempAlarm**”: Storage media has encountered a dangerous/damaging temperature threshold.

An example of one of the above is: “`/psialliance.org/Storage/mediaFailure/2`”.

24.0Appendix 9: “Metadata” Metadata Class Dictionary

Binary Tag (CRC32) = 0xB6625642

The following ‘Type’ fields/tags have been reserved within the ‘Metadata’ PSIA Metadata Class. This category of metadata indicates changes in the state of the metadata categories and/or source channels that are supported by a device or system. These dynamic changes are most likely to occur on Metadata/Event Proxies. The owning PSIA working group for this metadata class is the System group.

- “**update**” : A metadata/event update has occurred regarding metadata categories, and/or input channels, supported by a device.
- “**updatedCategories**”: This tag specifically identifies that the types of metadata categories offered by a device/system has changed.
- “**updatedChannels**”: This tag indicates that the input channel characteristics have changed. This usually occurs with the addition/activation or deletion/deactivation of input sources.
- “**marker**”: This type tag is used to indicate ‘no data present’ and act as a keep-alive synchromization marker in datastream mode sessions (HTTP, RTP, raw TCP/UDP). This event has no payload but a timestamp is included in the last field of the MIDS (past the LID field). Please see the example below.
- “**endOfMetadata**”: This payload-less event type is used to indicate, during a metadata/event stream, that the source has no more information. For HTTP/TCP-based metadata sessions, this allows the session initiator, or the source, close the session gracefully. This event has an ‘indicator’ field, following the LID field, that instructs the recipient of the recommended action to take. The values are:
    - “**close**”: This indicates that the session is closing.
    - “**resync**”: This is used to indicate to the consumer, for Simple Reliable Get sessions, that a GET needs to be re-issued to retrieve more data.
    - “**restart**”: This value indicates a new session needs to be established; the current one is disconnecting. A source also uses this value to gracefully disconnect sessions when they are unneeded/under-utilized.

Examples of some of the above are: “/psialliance.org/Metadata/updatedCategories”
“/psialliance.org/Metadata/marker/2012-12-31T09:53:07Z”
“/psialliance.org/Metadata/endOfMetadata/resync”

25.0 Appendix 10: “Network” Class Dictionary

Binary Tag (CRC32) =

The following ‘Type’ fields/tags have been reserved within the ‘Metadata’ PSIA Metadata Class. This category of metadata indicates changes in the state of the metadata categories and/or source channels that are supported by a device or system. These dynamic changes are most likely to occur on Metadata/Event Proxies. The owning PSIA working group for this metadata class is the Systems group.

- “**nodeUnreachable**” : A node (device, system, etc.) on a network is not ‘reachable’ (i.e. cannot be contacted or connected to). This error may be caused by an errant IP address, domain/host name, a blocked network path (i.e. Firewall boundary), or a network outage.
- “**connectionLost**”: An established network connection (i.e. socket) was disconnected unexpectedly.
- “**connectionRecovered**”: A prior, disconnected network connection (i.e. socket) has been re-established (see “connectionLost”).
- “**connectionActive**”: A network connection (i.e. socket) has been established.
- “**networkInactive**”: The physical network connection (MAC or PHY layer) is inactive.
- “**networkActive**”: The physical network connection (MAC or PHY layer) is active (see above).
- “**networkError**”: The network has encountered an error. This usually pertains to the MAC layer operations of a network.
- “**ipNetError**” An IP (DLC-layer) network error has occurred.
- “**networkStats**”: MAC layer statistics are being provided (e.g. RFC 2665).
- “**ipNetStats**”: IP network layer statistics are being provided (e.g. RFC 4293).
- “**networkOverrun**”: The MAC layer network interface has encountered traffic overrun. Network packets/frames have been lost.
- “**mtusLost**”: Connection/session layer loss of data (i.e. MTUs) has occurred.
- “**sessionActive**”: Event used as a session level keep-alive for metadata streaming sessions that have time gaps without information. This type does not have a payload; just the metaID (MIDS) and time.

Examples of some of the above are:

```
“/psialliance.org/Network/nodeUnreachable/1/10.5.13.204”,
“/psialliance.org/Network/connectionLost//216.74.33.9”,
“/psialliance.org/Network/networkOverrun/1”
“/psialliance.org/Network/sessionActive//2012-12-31T09:53:07Z”
```

26.0 Appendix 11: System Battery Events

All Battery-related Events are correlated to the '/PSIA/System/Battery' resource defined in the "PSIA Service Model v2.1" (or later) specification. The Class for batter events is "Battery"., and the only Type presently defined is "state". Battery "state" events pertain to changes in role and/or state and/or charge level. For more details please reference the '/PSIA/Batter' rsource definitions in the Service Model specification. The MIDS (MetaID Strings) root for all Battery state events is:

"/psialliance.org./Battery/state".

This prefix followed by the required ID field information. E.g.:

"/psialliance.org/Battery/state/2

Where the value "2" is the ID value for the respective battery component. Please reference the "PSIA Service Model v2.1" for more details on the System Battery resources.

The XSD file for Battery events is: "batteryEvents.xsd" and is listed in detail in Section 14.1.8 of the PSIA Service Model specification.