Physical Security Interoperability Alliance
PSIA Specification: Common Metadata/Event Formats and Transports
Version 1.1, Rev. 0.3b
December 7, 2010

Confidential Information for PSIA Use Only

PSIA Common Metadata/Event Management Specification
Version 1.1
Rev. 0.3b

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
Version 1.1 Rev 0.1	Created initial v1.1. spec from CMEM spec v1.0, Rev. 0.8b. - Added a new Section 9.6 to cover pure RTSP/RTP metadata streaming. - Updated Section 10.2.2 for new Version 1.1 "MetaSessionSupport" XSD that indicates support for querey string parameters or not, - Updated Section 10.2.4.1 to describe use of new query string paramters as part of stream session setup. All deltas to the v1.0 CMEM soec are in 'Purple'.	June 4, 2010 – June 7, 2010.	Roger Richter
Version 1.1, Rev. 0.2	Fixed minor typos from Rev 0.1; Added nnew Section 10.3 to cover Event Notification. This new service 'Actions' supercedes the Event notification currently outlined in IP Media Device Specification v1.1, Section 7.15, "/PSIA/Custom/Event".	June 30 – July 7, 2010	R. RIchter
Version 1.1, Rev 0.3	-Added Section 11 to describe how to detect CMEM services, and how to detect v1.1. support versus 1.0 support. - Updated External Document References - Merged in review comments. - Updated XSD files based on XSD to C# compilation compatibility; also, merged in the use of the new PSIA Common Types XSD file. -For each XSD, added filenames in the resource sections so that the XSD can be cut/copied directly into files for use. This could be done before, but the inclusion of XSDs with explicit file references would've made it a 'puzzle' to be solved by the user. - Updated Event Schedule XSD definition to allow multiple timespans per day. - From review, changed '/PSIA/Metadata/Actions/ capabilities' to '/PSIA/Metadata/Actions/attributes' to not conflict with the PSIA Service Model definition of 'capabilities' resources.	July 16 – August 3, 2010	R.Richter
Version 1.1, Rev. 0.3a	Added some new 'types' to the PSIA defined/reserved Metadata classes listed in the document's appendices. Added new 'Network' metadata class definitions (Appendix 10).	Aug. 8, 2010	R. Richter
Version 1.1, Revision 0.3b	- Updated embedded XSD files to match the PSIA schema repository copies. - Changed all GCMH *Streaming over HTTP* to use a new MIME multipart content type. Web tools prefer this method of content transfer vs a new definition. Updates made to Sections 9.2 and 9.3; also to Table 9.4.1. Transactional GMCH HTTP session remain unchanged. - Updated Section 10.2.4 to allow support for both GETs *and* POSTs for setting up session parameters. - Added new "sessionActive" type for the 'Network' class so that there can be a session level keepalive event.	Oct. 10/22, 2010	R.Richter
3	- Added Version attributes to key XSD objects and rev'd ALL version governed objects to version "1.1" irrespective of generation level to synchronize all items to this spec level. Changes are in RED.	Dec. 7, 2010	J.Wang/R.Richter

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

As with all standards initiatives, it is necessary to identify requirements. The following items are considered to be requirements for metadata and event processing in the PSIA system model which is projected to encompass Video, Access/Intrusion Control, Fire and Safety, Mass Notification, and Emergency Response types of systems, and components.

Basically, the above requirements breakdown into enabling:

A) The orderly coexistence of various forms of metadata in a multi-domain system
B) Support for a common processing model for all metadata and events
C) The ability to register/publish/subscribe to forms of metadata and event information dynamically
D) The ability to access metadata and event information in the appropriate method, flexibly (i.e. based on information types, network characteristics, and application needs)
E) The ability to meet high-rate/high-demand processing scenarios in the most cost effective manner possible
F) New capabilities that take PSIA standards well into the future.

2.1 Design Objectives

The outlined Requirements lead to the following Objectives (and design principles):

- Support for multi-domain event types and multiple event payload types (XML, binary, plain text, etc.)
- Efficient, real-time hierarchical classification & processing of Events
- Support for dynamic registration of event class/types by multiple systems (i.e. no code rebuilds and new versions for event updates)
- Flexible, extensible event/metadata design, including the ability to cross-correlate events/metadata.
- Inherent versioning of event formats (backward compatibility)
- Uniform event/metadata format such that event and metadata formats are the same for storage and messaging
- Efficient event design for generation, forwarding, searching and filtering
- Inherent support of event *priorities* for multi-domain and class differentiation
- Support for optional, variably sized, event objects (images, text, logs, etc.) as 'attachments'
- Transport independence and flexibility for 'push', 'pull', 'broadcast', and 'streaming' support.
- The ability to multiplex various forms of metadata across single connections

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

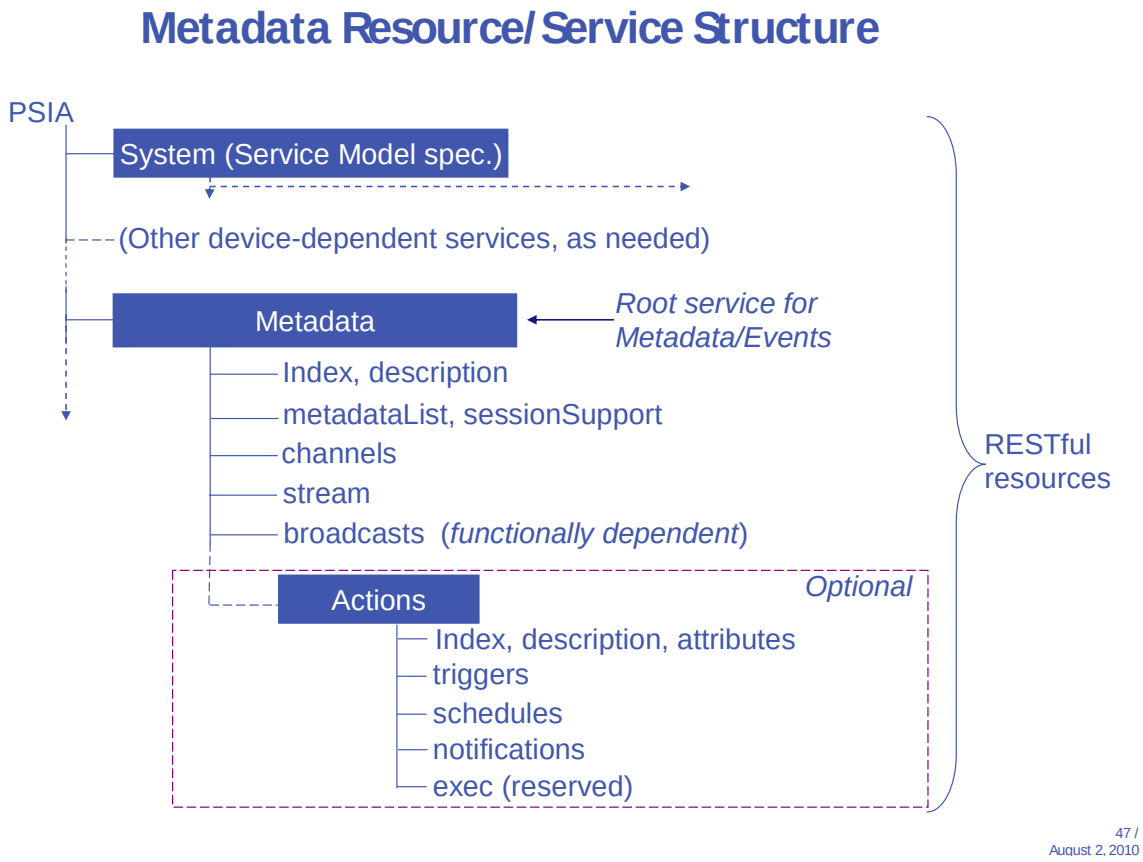
The CRUD operations are defined by the HTTP method as shown in the table below.

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



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**	**Mandatory/Optional**
description	Will respond to an HTTP GET with a <ResourceDescription> datablock	Mandatory
capabilities	Will respond to an HTTP GET with a resource-specific datablock	Optional
index	Will respond to an HTTP GET with a <ResourceList> datablock	Mandatory
indexr	Will respond to an HTTP GET with <ResourceList> datablock	Optional

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.

4.1 Overview of Metadata Resources

Name	Type	Description	Mandatory /Optional
Metadata	Service	Base Service resource for all the functional objects within the Metadata Service hierarchy.	Mandatory
index	resource	Required PSIA defined resource that lists the child-level resources within a service.	Mandatory
description	resource	Required PSIA-defined resource that describes the functional attributes of a service/resource.	Mandatory
metadataList	resource	Metadata resource that describes all of the active Metadata/Event types active on a particular device.	Mandatory
sessionSupport	resource	Metadata resource that defines all of the transport, format and session parameters offered by a device for transferring metadata information.	Mandatory
channels	resource	Metadata resource that contains all of the attributes and configuration information for all metadata/event input channels to a device or system.	Mandatory
stream	resource	Metadata resource that acts as the access point for creating metadata/event data streams.	Mandatory
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.	Optional/ Dependent
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.)	Optional (See next Section)

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

The '/PSIA/Metadata/Actions' Service is optional. 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 10.3*** 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	Mandatory/Optional/ Conditional
..Actions/index	resources	Required PSIA defined resource that lists the child-level resources within a service.	Required
..Actions/description	resource	Required PSIA-defined resource that describes the functional attributes of a service/resource.	Required
..Actions/attributes	resource	Actions-specific resource that describes functional limits and attributes of the Actions resources.	Required
..Actions/triggers and ..Actions/triggers/<ID>	resources	Definitions of conditions that be classified as events and thereby drive event notifications.	Required
..Actions/notifications and ..Actions/notifications/<ID>	resources	Definitions of specific notification methods that may be employed for certain event triggers.	Required
..Actions/schedules And ../Actions/schedules/<ID>	resources	Week-based calendar definitions that govern when 'triggers' are dormat versus active.	Optional

4.1.2 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.

4.2 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.

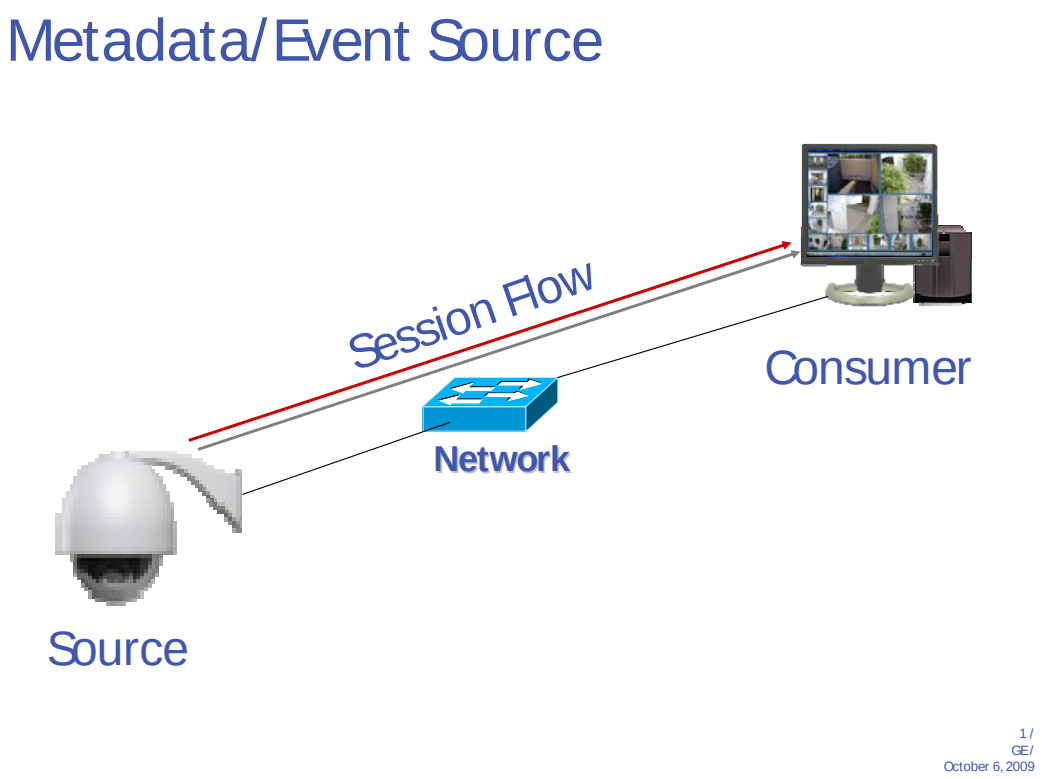
- **GET /PSIA/Metadata/metadataList**
  - *Gets the schema instance describing all of the metadata properties for the metadata domain/class/types active on a particular device or system.*
- **GET /PSIA/Metadata/Actions/index**
  - *Returns a schema instance with the metadata event signaling types (denoted 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.*

5.0 Metadata/Event Entities and Roles

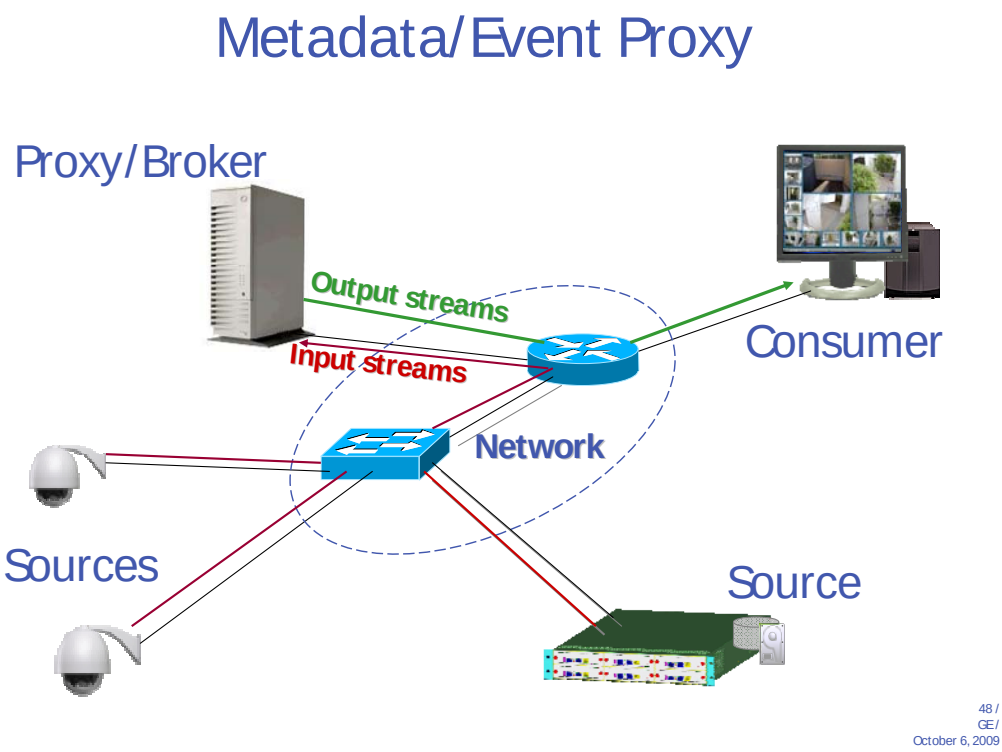
This PSIA specification covers the formats, protocols, and functional behaviors of PSIA compliant devices that deliver metadata and/or events. Operationally, there are 3 primary entities involved in the origination, delivery and consumption of metadata. These entities are:

- **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.

5.1 Simple Metadata Source Access Model



5.2 Proxy/Broker-based Metadata Access Model



6 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.

7 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.

7.1 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:

Table 7.1.1: 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.

7.2 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:

Table 7.2.1: 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 'positonal' 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:

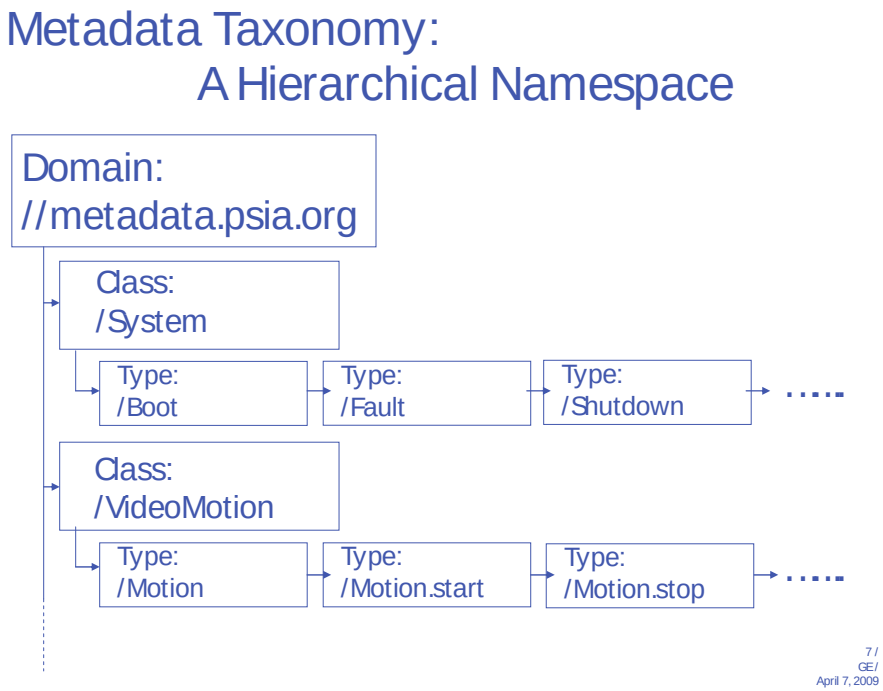
`/psialliance.org/Intrusion/alarm//C1EB2D39`

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 (though it is not encouraged). Any appended, after the ID field, is ignored by the common processing code and considered instance or manufacturer specific.

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

Figure 7.2.1: 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 in ***Section 8***. 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.'

7.2.2 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.

7.2.2.1 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.

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

7.2.2.2 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.

7.2.3 Metadata/Event Uniqueness

It is significant to note that metadata instances are unique from all other occurrences based 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.

7.3 *Time*

DATA TYPE: XML ‘dateTime’ (W3C format, ISO 8601 compliant), and NTP Time (GMCH)

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.

7.3.1 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.

7.4 *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:

Table 7.4.1: 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**

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.

7.5 *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. See ***Section 8.2***. for more guidelines.

8 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.

8.1 PSIA Metadata/Event Information Formats

Systems and devices in the PSIA protocol framework may select and support Metadata and Event information that is in one of two possible formats:

- **Classification/Encapsulated Format**: All sources must support the ability, at the request of a consumer, to provide their metadata/event information in a format that has a common, payload agnostic classification header. This header is called the General Metadata Classification Header (GMCH). Fundamentally, this header is a protocol independent header that is prefixed to the native metadata/event information issued by a device. A GMCH is an encapsulating header that is transport agnostic, and provides for real-time rules/filter processing. This format and its constructs are discussed in a following section.
- **XML Format**: For those devices where it is conducive, XML schemas/documents are encouraged as the format for conveying metadata/event information. Devices and systems that do not use XML documents for metadata/event information, must support the GMCH encapsulation method for indicating their payload type and structure.

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

8.2 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. The XSD for the XML ‘type’ that defines the “MetaHeader” is listed below.

Source 8.2.1: MetaHeader XSD (“metaHeader.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="1.1">

<xs:include schemaLocation=
"http://www.psialliance.org/schemas/system/1.0/psiaCommonTypes.xsd"/>

<xs:complexType name="metaHeader">
       <xs:sequence>
              <xs:element name="MetaVersion" minOccurs="1" maxOccurs="1"
                     type="xs:float">
                     <xs:annotation>
                            <xs:documentation> The version of the metadata header
                                   format
                                   should be "1.0" for the first revision
                            </xs:documentation>
                     </xs:annotation>
              </xs:element>
              <xs:element name="MetaID" minOccurs="1" maxOccurs="1" type="xs:anyURI"/>
              <xs:element name="MetaSourceID" minOccurs="1" maxOccurs="1"
                     type="GlobalID"/>
              <xs:element name="MetaSourceLocalID" minOccurs="1" maxOccurs="1"
                            type="LocalID">
                     <xs:annotation>
```

```
                                <xs:documentation> This field is the
                                        channel/track/zone/area/ROI/...
                                        ID, of the source, that corresponds to the
                                        associated metadata/event information in an
                                        occurrence; e.g., for
                                        Video Motion event the channel ID of the video input
                                        would be reported.
                                </xs:documentation>
                        </xs:annotation>
                </xs:element>
                <xs:element name="MetaTime" minOccurs="1" maxOccurs="1"
                        type="xs:dateTime"/>
                <xs:element name="MetaPriority" minOccurs="1" maxOccurs="1"
                                type="xs:unsignedByte"/>
                <xs:element name="MetaLink" minOccurs="0" maxOccurs="1" type="GlobalID"/>

                <!-- The following allows domain/category extensions ONLY after the std
                        header -->
                <xs:element name="MetaHdrExtension" minOccurs="0" maxOccurs="1"
                                type="HdrExtension"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="HdrExtension">
        <xs:sequence>
                <xs:annotation>
                        <xs:documentation xml:lang="en">
                                The following element MUST be a unique string that
                                identifies the ordaining body/group that has defined the
                                header extension(s) and the format thereof. The ordaining
                                body must publish/register its header extensions with the
                                PSIA on it public forum, or in its external document forum.
                                The format of the string is a URI where the first field
                                should identify the group that defines the header
                                extension; in many cases this is either a PSIA WG or a
                                mfgr. The following example is provided as a
                                guideline:"psia.SystemsWG/ExtendedReferenceIDs".
                        </xs:documentation>
                </xs:annotation>
                <xs:element name="ExtensionName" minOccurs="1" maxOccurs="1"
                        type="xs:anyURI"/>
                <xs:any namespace="##any" processContents="lax" minOccurs="0"
                                maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    </xs:schema>
```

The above 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.

8.3 General Metadata Classification Header (GMCH)

The processing objectives listed in Section 8.1 are based on real-world requirements. In order to satisfy these requirements, and provide additional capabilities, a general ‘Classification header’ definition is specified below.

Table 8.3.1: GMCH Field Definitions

Field	*Definition*
Signature: ASCIIZ[8]	Unique signature header (delimiter):'/GMCH\r\n\0' (allows HTTP/REST, raw TCP + UDP transports) also acts as a 'boundary' for MIME inclusion). ***Since this pattern acts as a signature field, and as a MIME boundary field, the tag must be ended with CR-LF and a NULL terminator; the pattern in hex is: "0x2F 0x47 0x4D 0x43 0x48 0x0D 0x0A 0x00"***
Version: Uint16	Version is 0x0100 (1.0) for this definition level
Priority: Uint8	8 priority levels (O is the lowest, 7 the highest = SysLog).
Structure/Type: Uint8	Simple or Complex structure indicator for instance info: (see following section for details)
Size: Uint32	Total size of all Event/Metadata information (inclusive)
Source ID: Uint8[16]	16-byte UUID/GUID of the Source
Link ID: Uint8[16]	16-byte UUID/GUID primary correlator of related event/metadata information.
Time: Uint64	NTP time of Event/Metadata occurrence
Source Local ID: Uint32	Source's numeric Channel/track/stream/area/zone ID, if applicable; if not applicable, it is set to 0xFFFFFFFF
Number Of Objects: Uint16	Number of objects that comprise this event/metadata instance
Event URI Length: Uint16	Size of Event's/Metadata's URI (see below)
Domain BTag: Uint32	CRC32 of URI's Domain ID/name (Section 4.3)
Class BTag: Uint32	CRC32 of URI's Class name (Section 4.3)
Type BTag: Uint32	CRC32 of Event's Type name (Section 4.3)
Event URI: UTF-8[ ]	Event URI String (padded out to 4 byte multiple)

The integer fields (e.g. Uint32, Uint64, Uint16) in the above header are all ***little-endian*** fields (i.e. native x86/ia32/ia64 format). Also the fields are clumped in 8-byte groups so that they are optimally aligned for 32- and 64-bit environments. The GMCH classification header is required, but SHOULD be supported by PSIA nodes that are metadata sources such that a consumer should be able to subscribe to the node's metadata and select that this header be imposed for all metadata instances. This header operates as an encapsulation header. I.e. it precedes and encapsulates one or more metadata objects that comprise a metadata instance. The following diagram depicts the simple relationship of this header with the potential metadata objects.

8.3.1 GMCH Structure/Type Indicator

Within the GMCH, the "Structure/Type" field indicates how a metadata/event instance, using the GMCH construct, is formed and how it behaves. The values for the "Structure/Type" field are:

- 0 = Simple Binary metadata/event instance

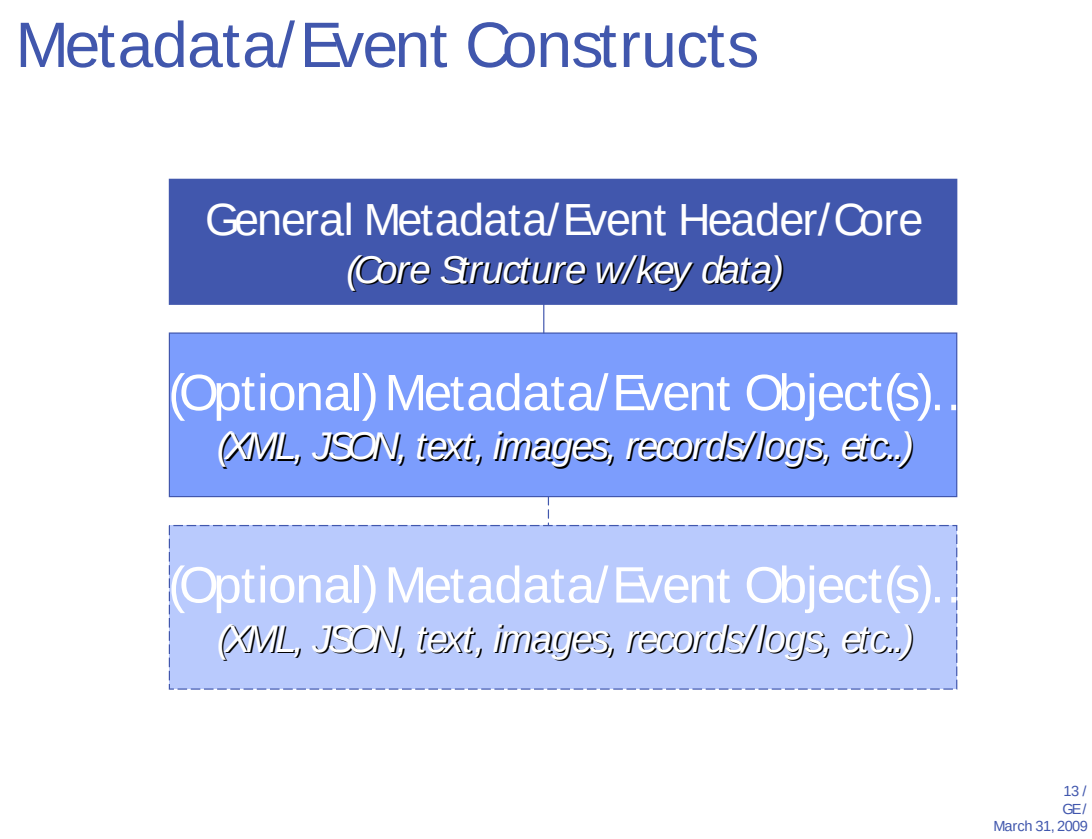
- 1 = Simple XML metadata/event instance
- 2 = Simple text metadata/event instance
- 3 = Complex (one or more objects) metadata/event instance
- All other values are reserved for future use

All of the "Simple…" formats are one contiguous structure with the GMCH directly preceding the data payload. The "Complex" format indicates that the metadata/event instance has an "object" succeeding the GMCH. The "ACK" indicates that the metadata/event instance is a standalone GMCH that contains the MIDS, the MIDS tags, the timestamp, and other GMCH fields of a prior metadata/event occurrence that is being explicitly acknowledged at the transaction layer.

More information on the structure of various GMCH types is detailed in the next section.

8.3.2 GMCH Metadata/Event Structure

Figure 8.3.1: Event Structure (GMCH / MOH Object Relationship)



The General Metadata Classification Header (GMCH) can also double as a small-sized, self-contained event instance for lower-end embedded applications such as control panels, etc. This header concept provides the following benefits:

- **Version Insensitivity:** By having common header that utilizes the Domain/Class/Type concept, any version of any form of information can be commonly processed irrespective of the format and revision involved. All additional information is basically a 'payload' succeeding the header (e.g. the Point-of-Sale text would be the payload).
- **Speed and Efficiency:** By converting the URI metadata fields, Domain, Class, and Type, to binary, fixed length tags, extremely efficient filtering, forwarding and searching is possible within a 3-level hierarchy, if needed. This enables processing thousands of events per second at fraction of the compute load that free-form, variable length string processing requires. This lowers cost and latencies.
- **Information and Format Flexibility:** The use of the GMCH enables any form, or even multiple forms, of information to be processed within a common framework. Current definitions in XML format are supported as payloads, or content, with respect to the GMCH, just like text information from PoS or ATM metadata sources. The support for legacy, and future, data definitions is inherently present, also. And, multiple formats can be processed simultaneously in a system employing this header framework allowing migration and legacy support, where needed.
- **Transport Independence:** Unlike many current event/metadata schemes, which rely heavily on HTTP/SOAP or HTTP/REST as the base protocol, this scheme is agnostic to the transport type, and control protocol, used to transfer metadata/event information. The use of TCP and UDP, in unicast and multicast modes, is inherently enabled. This is very useful in low latency, multi-consumer systems that use multicast UDP as a transport. Additionally, the GMCH allows multiple metadata/event instances to be sent in a single payload, or as a continuous stream. This is an area where XML documents have some transport dependencies.
- **Wide Applicability:** Due to the format and relatively small size of the GMCH, the same format used to construct a metadata/event message, can be used to store the metadata/event information without any modifications. It works as a message payload and as a stored metadata record. Additionally, the GMCH can be used by small embedded devices and large applications and systems, alike.

Metadata Object Header (MOH)

In addition to the above GMCH structure, each metadata object that accompanies a metadata instance MUST have the following Metadata Object Header (MOH):

Table 8.3.2: MOH Field Definitions

Field	***Definition***
Object Size: Uint32	***Size of entire object inclusive of MOH header***
MIME String Size:Uint16	***Size of the following MIME string inclusive of any padding characters (see below)***

MIME String Object Identifier: UTF-8[ ]	***MIME string identifying object payload type. This includes the "Content Type:" prefix. This MIME string MUST end with '\r\n\0' (CRLF-NULL; RFC 2045) followed by any padding [NULLs] necessary to align payload (mainly for image objects) on a 32-bit boundary. Text/XML payloads, etc., need no alignment.***

The MOH enables the inclusion of any set of relevant objects to be seamlessly included in a metadata/event instance information base. The following diagrams depict some example metadata/event references:

Figure 8.3.2: Reference GMCH Metadata Example (Simple Event)

Simple/Basic Metadata Instance:

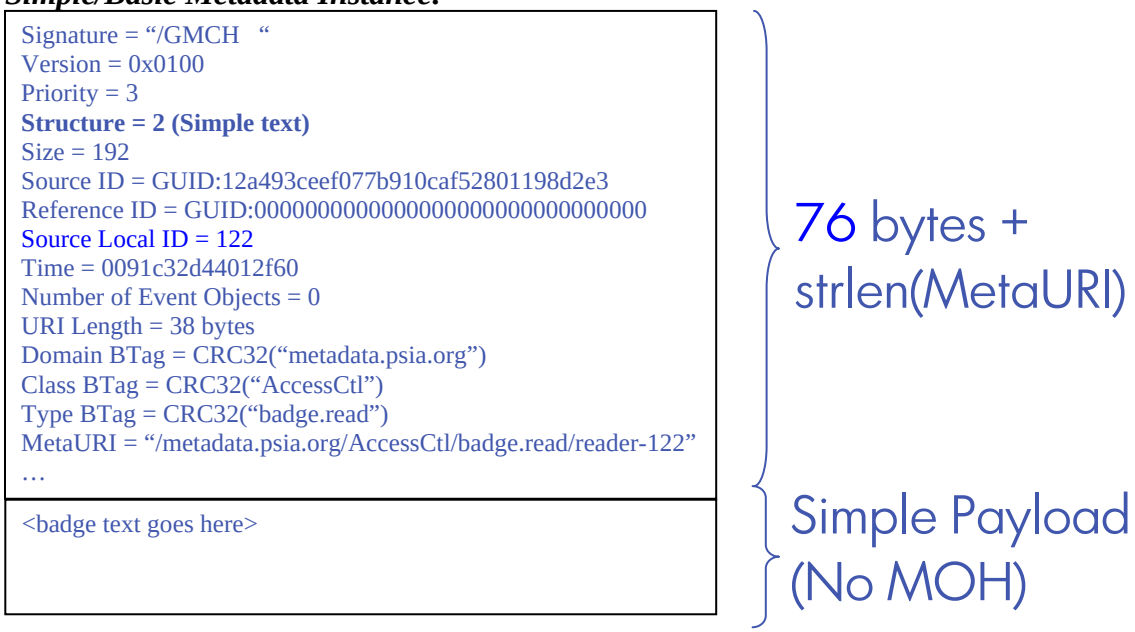
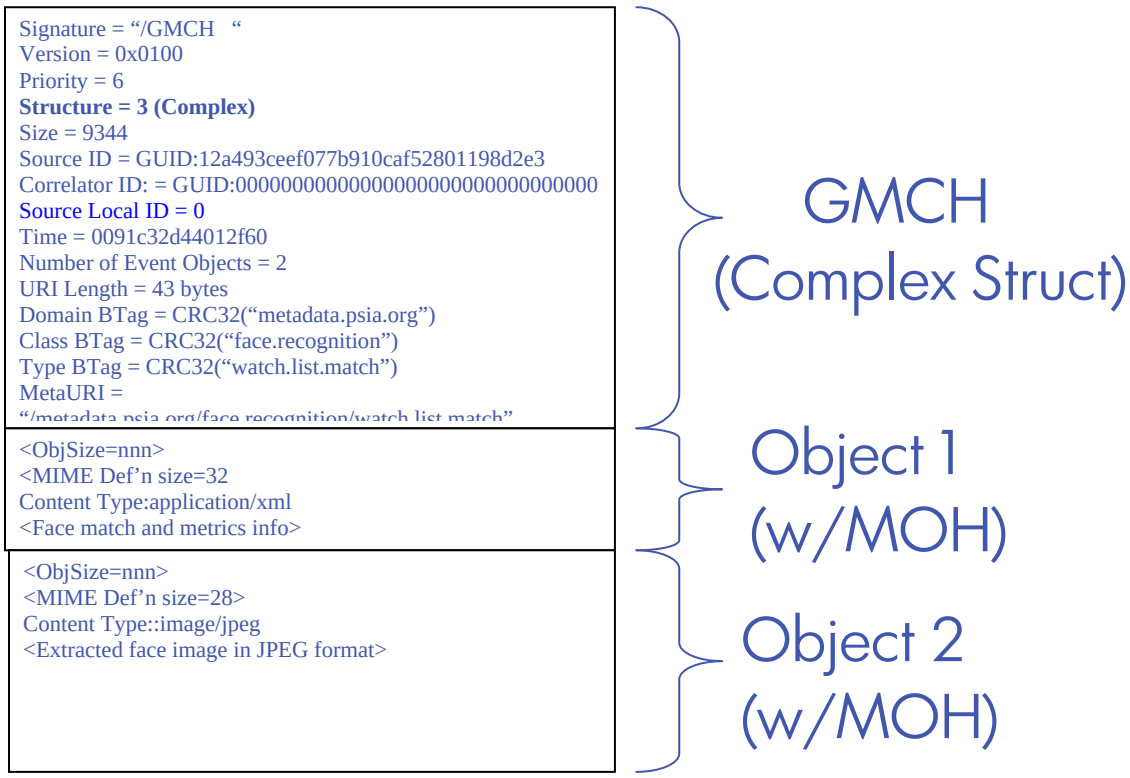


Figure 8.3.4: Reference GMCH Metadata Example (Complex Event)

Complex/Multi-Object Metadata Instance:



8.3.2.1 Use of the General Metadata Classification Header (GMCH)

The use of a common classification header has many advantages. As a PSIA construct it would be utilized to encapsulate the XML defined metadata issued by the relevant PSIA working groups. For example, the current event and metadata definitions set forth by the PSIA Analytics Working Group (AWG) would contain the common Metadata fields in XML as outlined in ***Section 3,*** along with the other currently defined alert/count related XML information. When a consumer/client application retrieved the description of the "Events" from the Analytics application, the application would indicate that it can provide the information in raw XML format, or encapsulated in a GMCH data wrapper. If the consumer selects the GMCH format, the standard XML definition is transferred to the consumer with a prepended GMCH structure. All additional and subsequent metadata definitions would be published as their own independent schemas, as is currently practiced in PSIA, though the sources would offer both an encapsulated version (i.e. with a GMCH prefix), and a non-encapsulated version.

8.3.2.1.1 GMCH Extensions and Limitations

As noted in prior sections, the GMCH provides 3 primary benefits:

- Enable a fast, efficient processing equivalent encapsulation header of the MetaHeader; and…
- Provide a common encapsulation header that is payload agnostic and allows multi-object composition; and…
- Provide a transport and session agnostic header that readily adapts to any necessary transport method.

Bullet number 1, in the above group, outlines that the GMCH and the MetaHeader XML definitions are designed to be synonymous. However, unlike the MetaHeader definition (see ***Section 8.2***), the GMCH structure does not allow extension fields/elements to the general header. Therefore, entities sanctioning extensions to the MetaHeader definition, such as additional Link IDs for multi-group correlation, must implement one of the following options:

- Specify and place additional fields in the trailing fields of the MIDS for the metadata/event categories that have need of additional extension information. Please note that the MIDS required fields are at the start of the MIDS and additional fields are allowed after the required fields have been supplied, or 'null' slotted.
- Specify and define a GMCH extension Object, using an MOH definition, that would trail, as an additional metadata instance object, the GMCH.

Either of the above 2 options are acceptable along with a combination of both (if desired). Basically, the key MetaHeader information must be distilled into the common GMCH information. Extension information, when needed, must be defined by the designated Working Group, using one of the 2 above options.

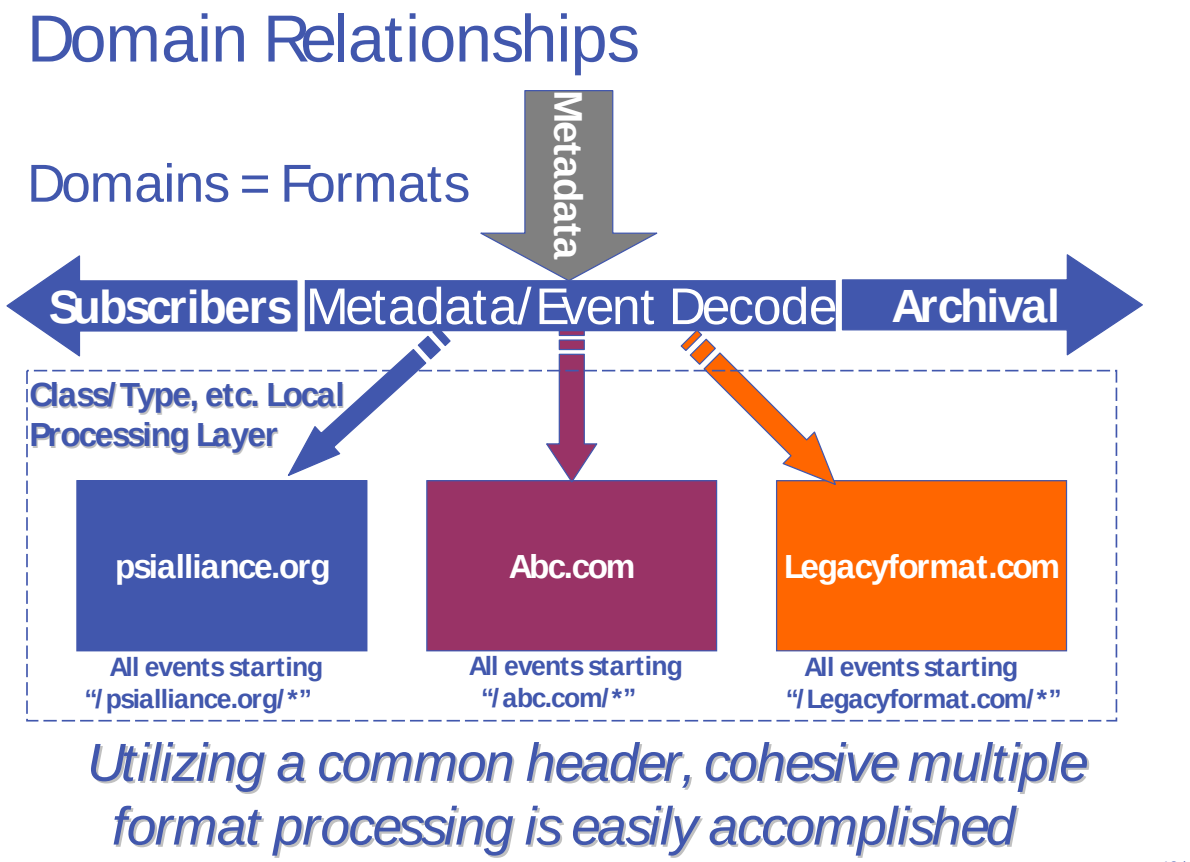
8.3.2.1.2 Use and Conversion of the Metadata Identity String (MIDS)

For systems that need to process standards compliant metadata, and still support other versions and formats, the GMCH enables all metadata information to uniformly, and cohesively, coexist in a given system (see ***Figure 8.3.5*** following).

As is evident, the MIDS' URI construct that identifies metadata and event classes and types, provides great flexibility in enabling publishers to create cohesive and extensible metadata definitions. It also eliminates the likelihood of namespace collisions due to the hierarchical compartmentalization of domains, classes and types. However, string comparisons are roughly %400+ more inefficient than integer comparisons. As such, the GMCH has 3 fast look-up fields for the Domain, Class and Type strings in the MIDS. These fields are 32-bit unsigned integers that contain the CRC32 derived value of each of these Domain, Class and Type string values. The CRC32 algorithm is bit position sensitive (which means it is case sensitive) thus rendering a strongly unique derivative value for each of the values in the 3-part tuple made up by Domain/Class/Type. This enables extremely efficient filtering, classification, and searching without losing the benefits of the URI nature of the MIDS. Source code is provided in ***Appendix A*** of this document for the CRC32 algorithm.

The Domain field of a MIDS' URI is the top-level identifier as to the ordaining body associated with the rest of the metadata information, and, therefore, the format designator. In other words, whoever creates the metadata/event definitions is directly associated with that format set. Consider the diagram below.

Figure 8.3.5: Domain Relationships



This diagram depicts the ability to have multiple, even dynamically loadable, format processors in a system utilizing the Domain IDs as the differentiator between the metadata/event processors. This enables future extensibility while allowing backwards compatibility. This model lends itself to environments where metadata format processing modules can load, register their domain tags with the metadata/event front-end decoder and process any metadata/event instances that match their respective domain types.

9 Metadata/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)
- **Asynchronous Reliable Notification Model:** Source is responsible to establish a session to one, or more, Consumers when an event occurs. One socket per consumer (TCP-based Push model).
- **Stream Model:** Consumer desires 'stream' of metadata information. The Consumer requests an RTP stream of metadata information from the Source. (UDP or TCP-based, requested Push model).
- **Group/Mass Notification Model:** Source issues metadata real-time on a multicast address. Multiple Consumers ('listeners') subscribe to the multicast sessions/channels. Low latency, highly scalable model for large systems. (UDP-based, multicast Push model).

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

9.1 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.

9.1.1 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 setp parameter.

Table 9.1.1: Transport Information Elements

Element Name	Source	Consumer	Description
‘metaFormat’	Required		This element defines which formats are supported for metadata/event transfer. The 2 choices are: • “xml-psia”: This represents that the format is the XML/XSD definitions, with the common meta-header information, published by the PSIA working groups. • “gmch-psia”: This indicates that the format is the General Metadata Classification Header (GMCH) format described herein. A metadata source MUST advertise which of the above are available when the “/Metadata/sessionSupport” resource is read. ALL metadata/event sources MUST support GMCH encapsulation; XML formats are optional since other formats must be GMCH formatted. Consumers MUST choose from the options presented by the source.

‘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.
‘metaSessionType:: metaSessionFlowType’	Optional	The ‘metaSessionFlowType’ is an optional field. Its use is dependent upon the protocol mode (see above) chosen for transporting metadata information. If one of the HTTP/REST based protocols is chosen, then one of the following 2 flow modes must be selected: • “datastream”: Sources MUST support this mode. 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 optional flow mode requires the consumer to issue a ‘GET’ for all data to be received from a source. 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.

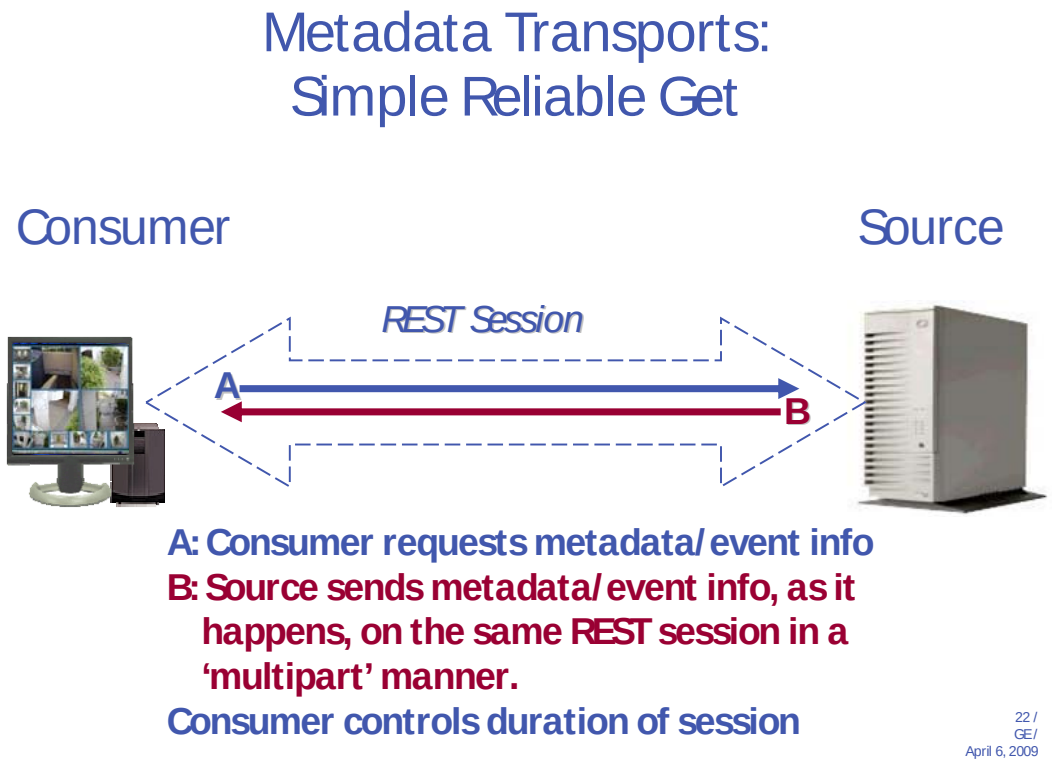
“metaSessionType”:: “metaSessionPersistent”	Dependent/ Optional	N/A	This dependent/optional element indicates whether sources can retain asynchronous HTTP/REST session parameters across reboots, or not. The default support level for basic devices is ‘False’. However, Proxy devices MUST indicate whether 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.
‘metadataNameList’	Dependent	Optional	For sources, this element only has validity

‘metaChannelList’	N/A	Optional	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. 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.

9.2 Simple Reliable Get Model (“RESTSyncSessionTargetSend”)

The Simple Reliable Get Model is a straight-forward 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.

Figure 9.2.1: Simple Reliable Get Overview Diagram



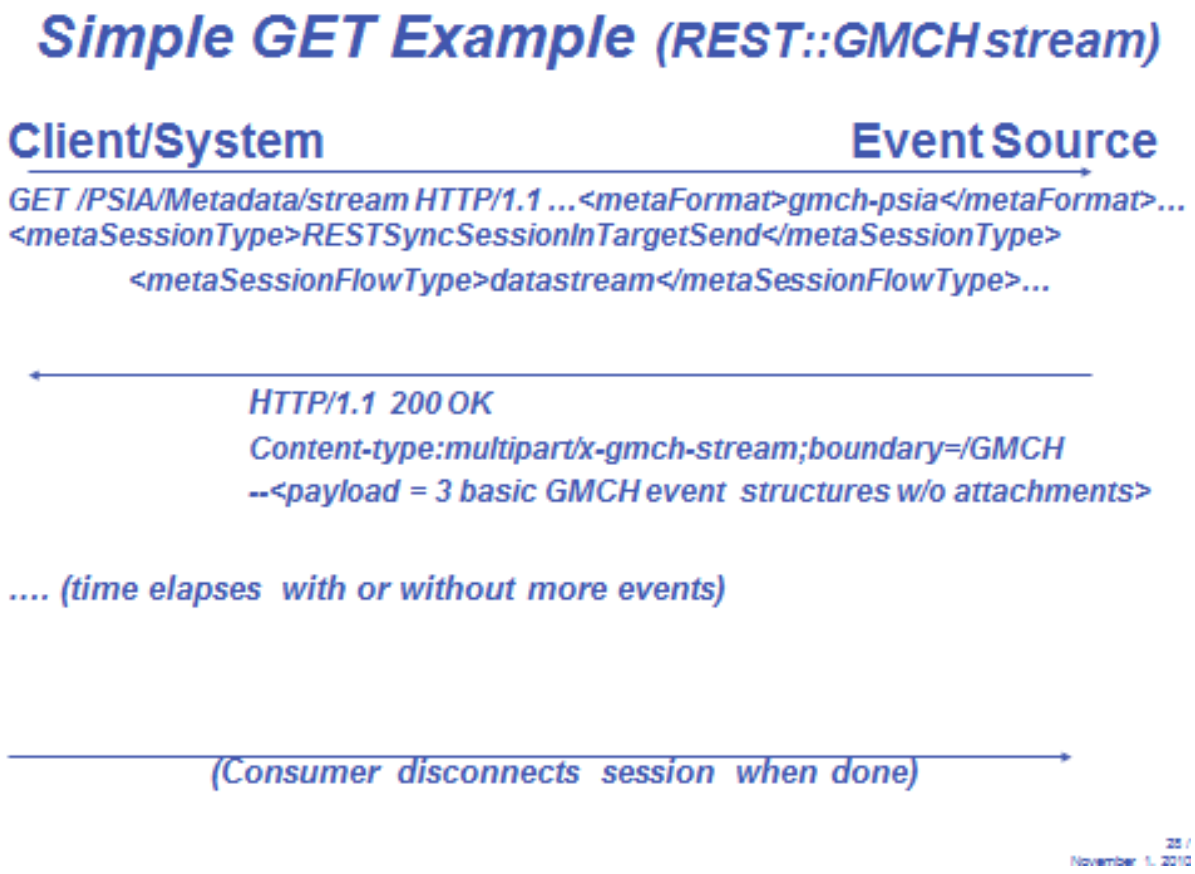
More detailed exemplary message flows are described below for the Simple Reliable Get Model.

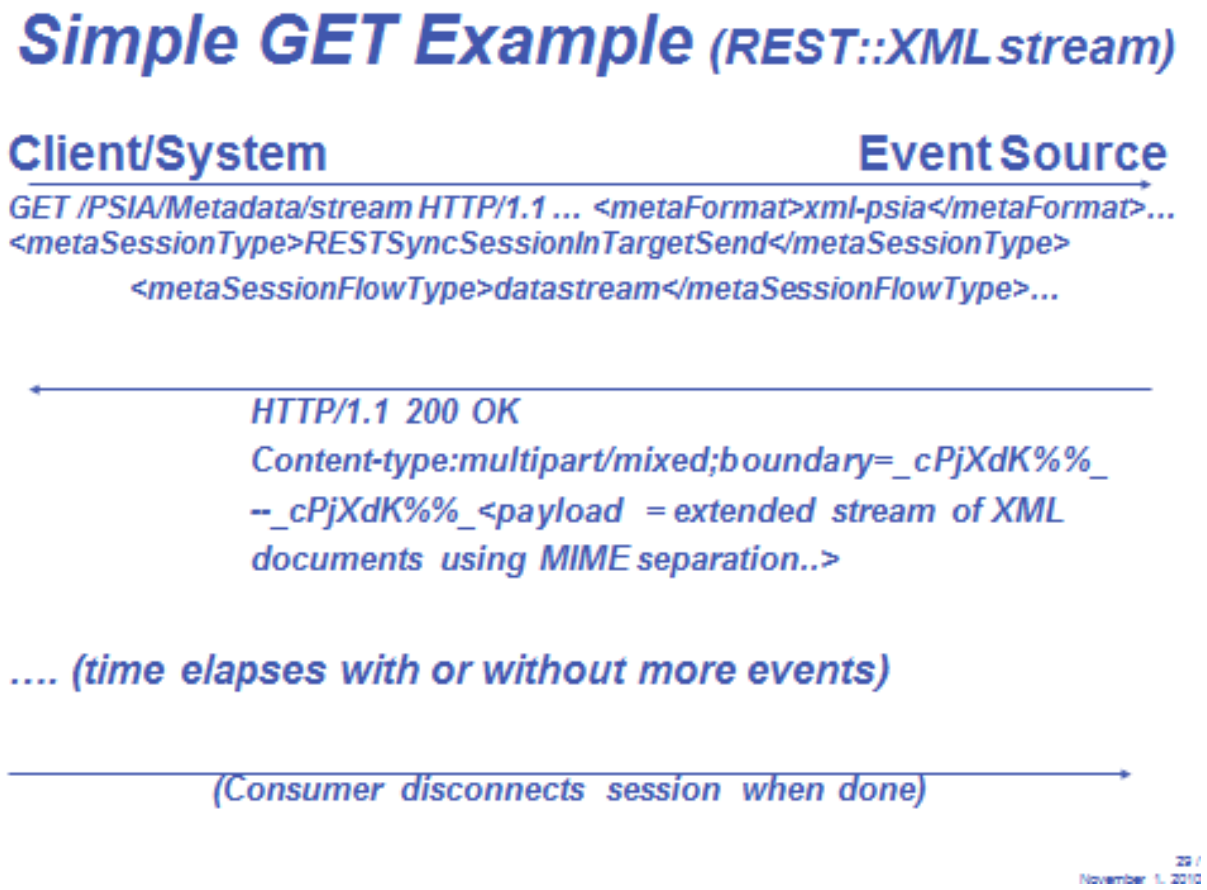
Figures 9.2.2: Simple Reliable Get (Message Flow Examples)

The following message flow diagrams cover the 4 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 following 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:

- 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 ***Section 10.2.4*** of this document.



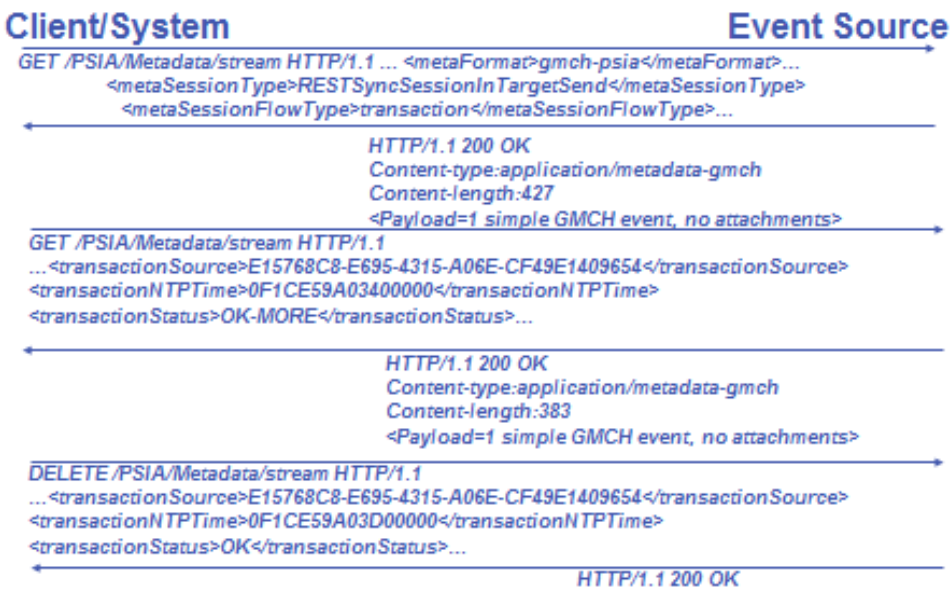


In the above XML example, the data stream is comprised of XML document instances. Therefore, the content type is "multipart/mixed" with a MIME boundary marker. Please see RFCs 2616/2387 for more details. The first session is GMCH formatted where each stream element is self defining and *only* requires the *starting* MIME multipart ("multipart/x-gmch-stream") transport wrapper with an initial boundary definition. This is because a GMCH stream is treated as one giant, multi-element document where only the starting boundaries are used for delimiting each element.

Figures 9.2.3 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; this is described in detail in ***Section 9.4.1***. For GMCH formatted information, ALL transaction acknowledgements are acknowledged using the NTP time ("transactionNTPTime") from the GMCH header ('sprintf(str, "%x", value)' equivalent) of the original event. 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 in ***Section 9.4.1*** for details.

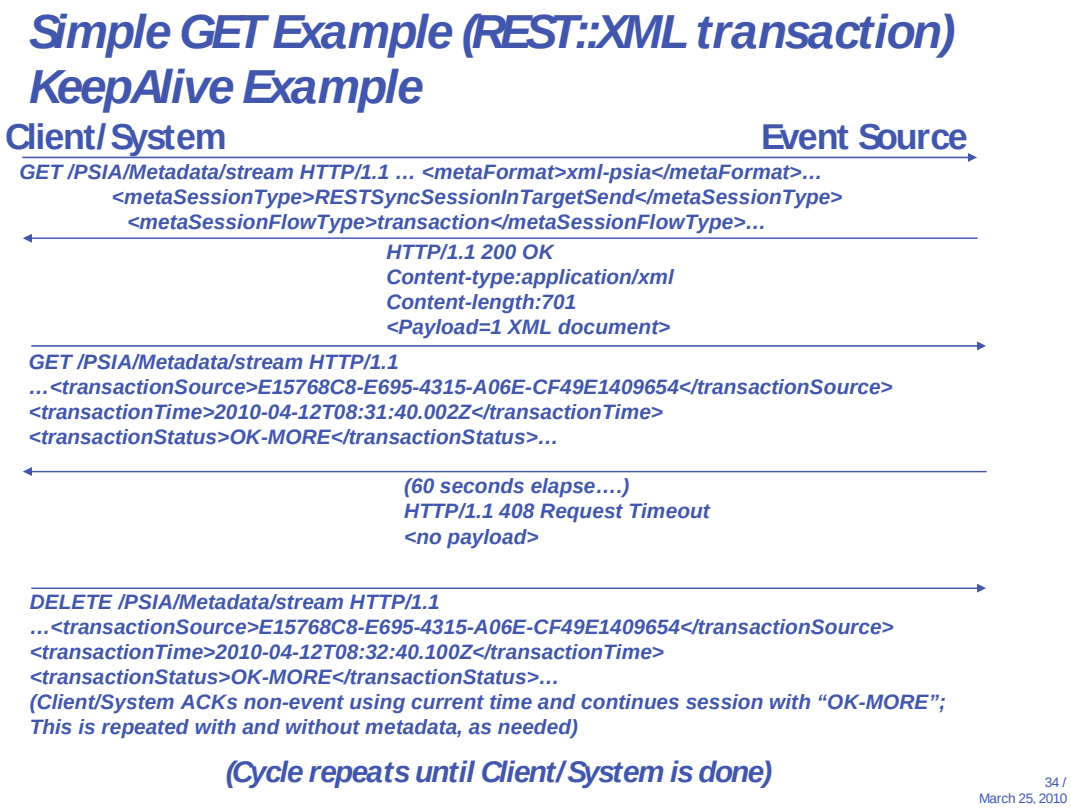
Simple GET Example (REST::GMCH transaction)



(Client/System disconnects session)

Simple GET Example (REST::XML transaction)

(Client/System disconnects session)



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.

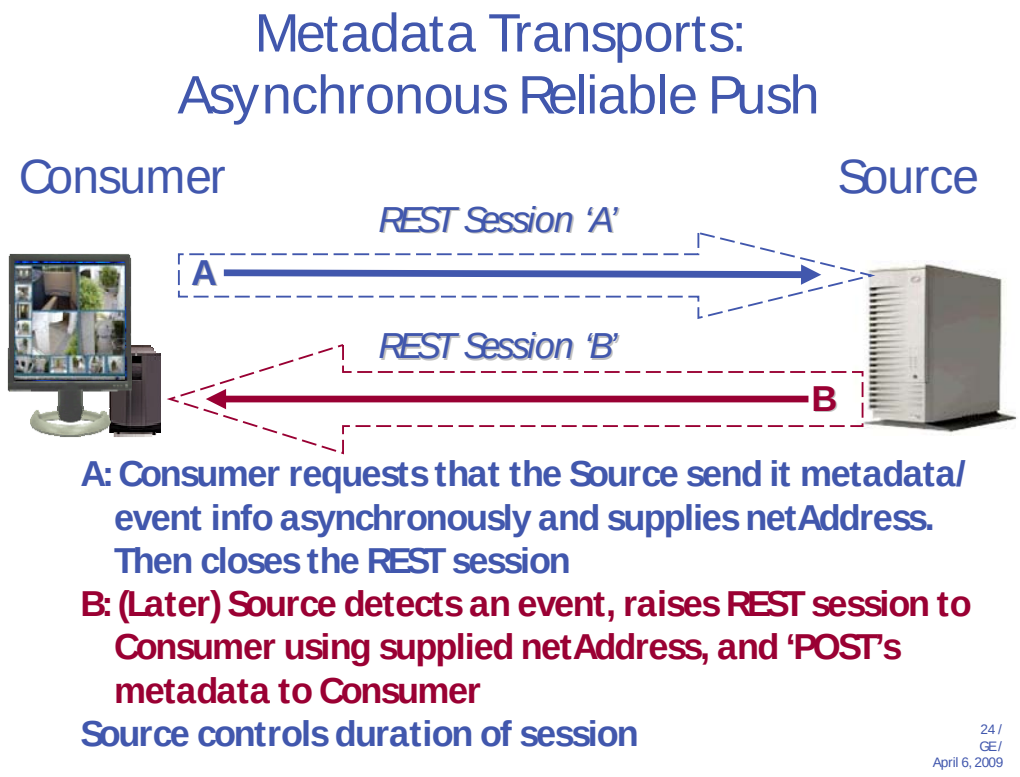
9.2.1 HTTP/REST Session Authentication

All HTTP/REST sessions may require session level authentication and/or session level security. Per the PSIA Service Model specification, ***Sections 4.3*** and ***4.4***, *all* PSIA devices must support both Basic and Digest level authentication for HTTP (RFC 2617). Support for HTTPS, in its various flavors is optional, but highly recommended for all devices. The session flow examples above do not show session level authentication in order to save space, and due to the fact that examples are listed in the PSIA Service Model specification. In all cases for the examples, session authentication would precede the message exchanges outlined in the flows.

9.3 Asynchronous Reliable Notification Model ("RESTAsyncSessionBackSourceSend")

The Asynchronous Reliable Notification Model can be an event, time, or reverse connection driven model. It has 2 basic benefits: A) in event-driven mode, no connections are established until there is data to be transferred (which aids scalability in large systems), and B) it allows redirection in that a metadata/event source can be directed to send its metadata/event information to any 'listening' system or device; this works well for environments that have metadata/event management proxies/brokers for systemic information processing. This model also obviates the need for setting up long lived connections, or using polled 'get' modes. The negative aspects are that: A) additionl latencies and errors are more likely in establishing asynchronous sessions at the time of an event, and B) there is some additional complexity in managing the session parameters for asynchronous connections for the duration of their lifespan. A simple depiction of this model follows.

Figure 9.3.1: Asynchronous Reliable Push Overview Diagram

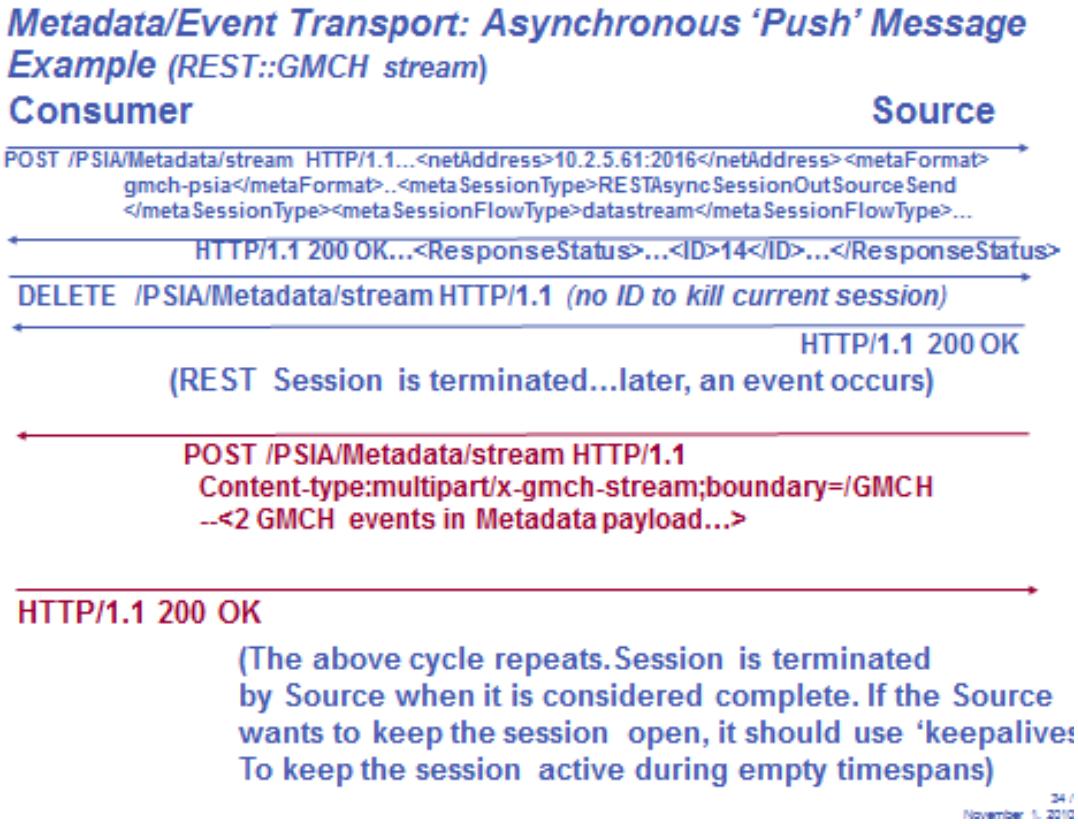


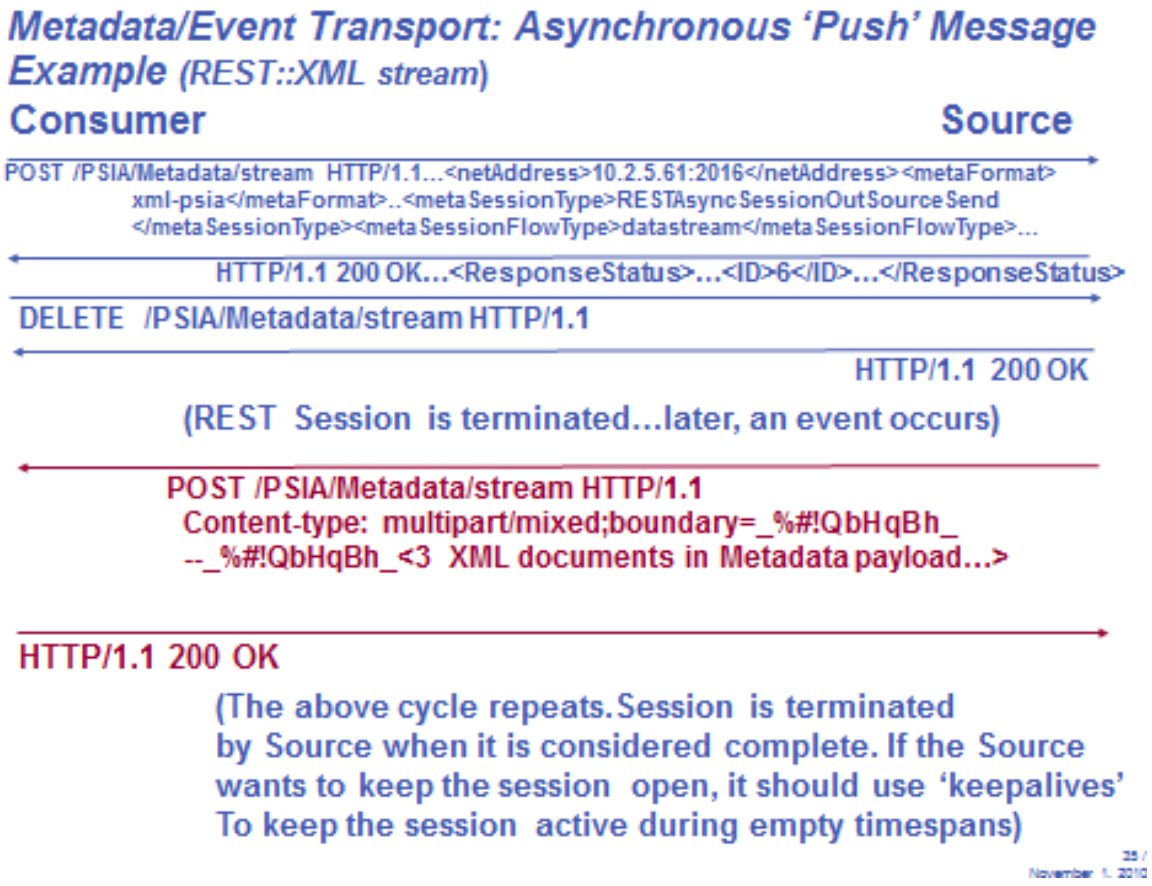
More detailed diagrams of the various message flows follow for the sessions using the following formats and flow types:

- GMCH and XML formatted information in 'datastream' flow mode;
- GMCH and XML formatted information in 'transaction' flow mode.

Please note that the setup of an asynchronous HTTP/REST notification session actually creates a REST resource with a unique ID (see ***Section 9.3.1*** for more details) and requires the use of the HTTP POST method. This is a key detail evident in the example message flows that follow.

Figures 9.3.2: Asynchronous Reliable Push (Stream Flow Examples)

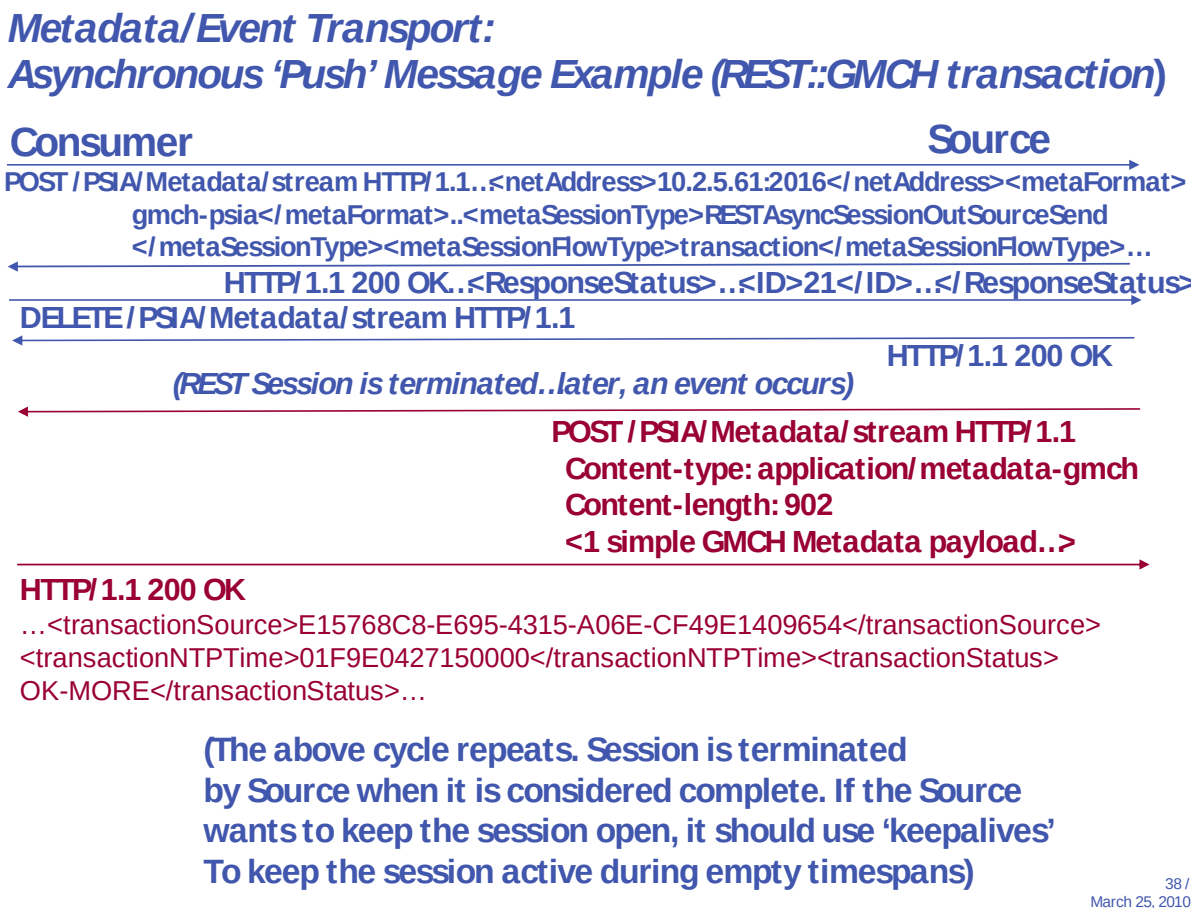




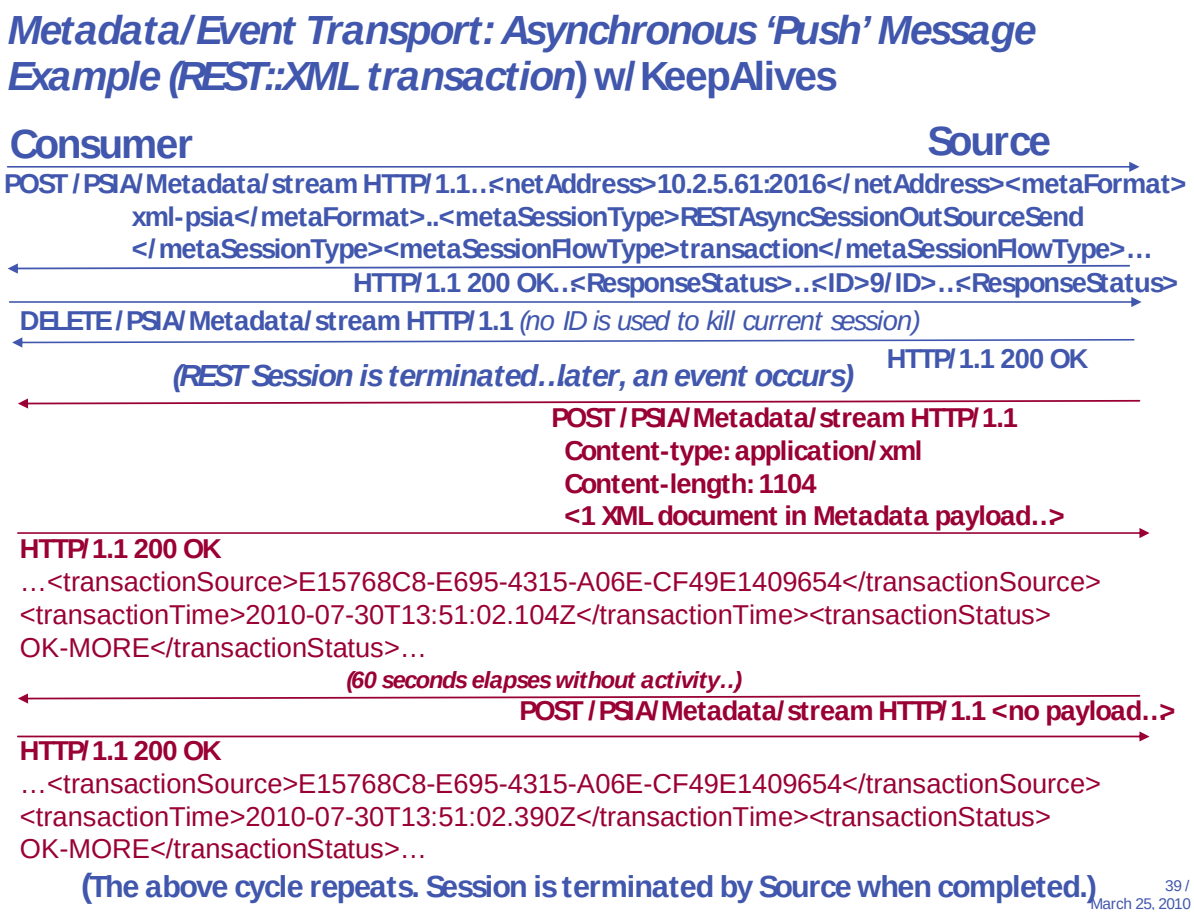
Please note that in the two above examples the notification session initiator, which is the metadata source, is responsible for the session's initiation, *and termination*. Additionally, this model provides the equivalent of a reliable stream model (though at a higher overhead premium). *Also, this model, like the other models following this one, allows the initial REST management session to continue for management purposes whereas the first model, Simple Reliable Get, monopolizes the initial REST session, for data transfer, thus requiring an additional REST session if the Consumer desires to perform any concurrent management operations (i.e. configuration, status, statistics, etc.).*

Figures 9.3.3 Asynchronous Reliable Push Transaction Flows

The next 2 message flow examples cover transaction flow modes.



These transactional scenarios require the consumer to 'ACK' each data chunk before the source will proceed with new data. The ACKs are always done with an HTTP/REST response and a payload containing an instance of the "MetaTransactionResponse" schema (see ***Section 9.4.1***).

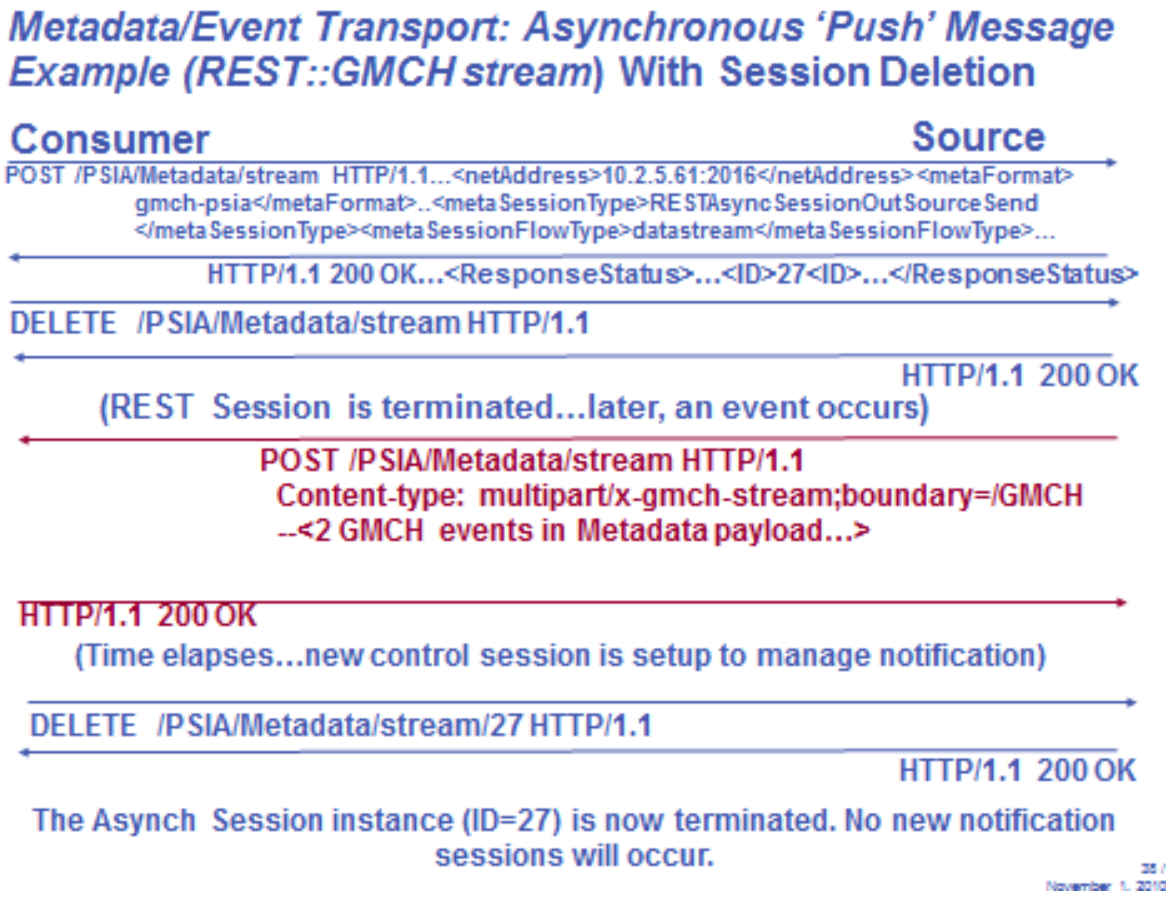


The above example also carries an example of an application layer 'keepalive' mechanism. The source sends data, when it occurs. However, after durations of inactivity, the source sends null HTTP/REST messages, without payloads, to keep the connection alive, as needed to ensure the health of the connection and that of the consumer.

9.3.1 Asynchronous HTTP/REST Session Management

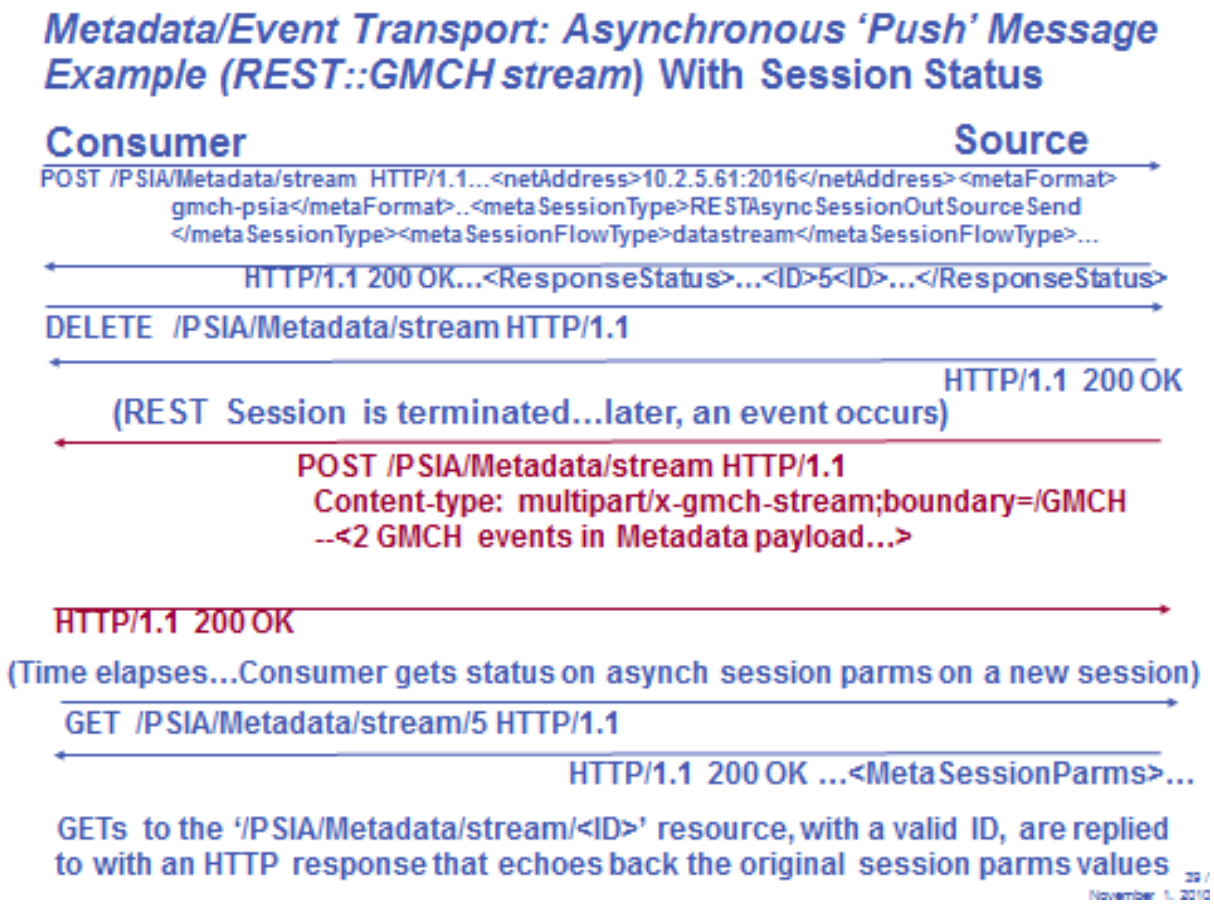
Asynchronous HTTP/REST sessions are managed differently than the synchronous session types. This session management difference is related to the fact that the session parameters, and the ability to trigger asynchronous connections, persist after the initial session has set the apprrpriate session parameters. As such, asynchronous sessions are treated as persistent resources once they have been established. In compliance with the PSIA Service Model specification v1.0, Sections 4.12 and 10.1.4, all asynchronous sessions are created via 'POSTs', and IDs are generated for each session instance by the metadata source. The session instance IDs are returned to the session initiator in the 'ResponseStatus' schema instance that is part of the HTTP response message stimulated by POST. So, unlike the 'Simple Get Mode' which uses GETs to start session activation, POSTs are used to created session instances which must eventually be deleted. Therefore, a source device MUST assign a device unique ID to each successfully created asynchronous session instance. This stream ID basically becomes the 'handle' for the session's unique parameter set. Devices must choose whether they support

persistent copies of the asynchronous session parameters, or not. The default support level is that the session parameters are not persistent. Only Proxy nodes ***should*** support asynchronous session parameter persistence. Any device can optionally support the persistence of session parameters across reboots.
The following message flow example depicts the setup and deletion of an asynchronous notification session.



In the above example, the session initiator establishes an asynchronous session instance which the source assigns the ID of '27.' This causes one, or more, asynchronous notification session instances to occur. At some point in time, the session initiator decides to cease receiving notifications and uses the DELETE, with the originally assigned stream ID (e.g. DELETE /PSIA/Metadata/stream/27), to permanently delete the session parameters.

Since asynchronous sessions, once created, 'exist' until deletion, which depends on a device's support for persistence, a GET can be issued against a respective stream's ID to determine the state of that session's parameter set. All GETs to a 'stream ID' (see ***Section 10.2.4***) must respond in the HTTP response message with the 'MetaSessionParms' instance object associated with that session resource, if the session parameters are still intact. This allows a level of state management for asynchronous session instances/parameters. The following message flow depicts an example exchange.



To summarize, ALL asynchronous HTTP/REST sessions are created via the HTTP POST method, are given device unique IDs at the time of creation, and are required to be explicity deleted, via the stream ID, by the session initiator. Proxy devices should maintain asynchronous session instances/parameters across reboots. Details regarding the parameters, and schema definitions are described in ***Section 10.2***.

9.3.2 Asynchronous HTTP/REST Authentication

Out of all the session types that may be employed for the transfer of metadata information, the asynchronous HTTP/REST session types are unique in the fact that they *may* require some level of session authentication for the subsequent notification session(s). In all of the above example session flows, session authentication has been omitted from the examples due to its complexity (i.e. the amount of space the login information requires for description). However, if an asynchronous notification session requires authentication, the consumer must supply the session authentication information as part of the session setup parameters. This is covered, in detail, in ***Section 10.2.4***. Please note that, in all of the session flow examples, session authentication would precede, but not alter, the message flows described in them.

9.4 HTTP/REST Protocol Flow Design

The above Sections of this document cover the design of HTTP/REST sessions as the transports for metadata/event information. Since metadata/event information can be offered in either XML, or General Metadata Classification Header (GMCH), format, the rules for transporting the data within an HTTP message payload vary based on the format of the metadata/event payload type. Additionally, some systems, or devices, may require that the transfer of metadata/event information be performed in a transactional manner based on the device and system type managing event data. This affects the dynamics and mechanics of managing the information flows. Especially when HTTP, by nature, is a half-duplex oriented protocol. The following table outlines some of the key details, and differences, in the combinations of formats and flow types.

Table 9.4.1 HTTP/REST Formats and Flow Types

Format	Flow Type	Content Type	Notes
GMCH	datastream	" multipart/x-gmch-stream; boundary="/GMCH " "	REQUIRED: Source sends data in a continuous stream. Consumer cannot 'ack' the data; it only wants to reliably receive it. Streams are cancelled via session disconnection. If a source closes an HTTP message (i.e. a stream chunk) the consumer MUST reissue a GET and the source MUST assume that a new GET is imminent and NOT lose any data. No Content-length field is provided and the '/GMCH\r\n\0' signature, in the GMCH header acts as the multipart boundary marker (with the preceding, required "- -" double-dash character markers).
XML	datastream	"multipart/mixed"	OPTIONAL: Same as a GMCH stream except that: • HTTP/MIME content type is different; • No content length is needed; • Boundary definitions and scanning are required, which have increased overhead. In this mode XML document instances are transmitted 'back-to-back in the stream, irrespective of time gaps (i.e. the data stream syntax is the same even if there is a time gap between events). The same rules as apply as the GMCH

			datastream mode regarding stream management.
GMCH	transaction	“application/metadata-gmch”	OPTIONAL: 1-for-1 transfer/ack of metadata/event information encapsulated in GMCH format. All received metadata instances are acknowledged in HTTP messages with XML payloads that contain a “MetaTransactionResponse” schema instance. The flow and status of the information exchange are governed via the “transactionStatus” field in the response information.
XML	transaction	“application/xml”	OPTIONAL: Same as the above GMCH transaction information except that the metadata instance information is in XML document format.

Application level ‘keepalives’ are performed in both ‘Pull’ and ‘Push’ modes by the metadata/event source sending HTTP/REST messages without any payloads. Consumers receiving HTTP/REST messages without payloads on active sessions, MUST respond with HTTP/REST messages containing a “MetaTransactionResponse” schema instance containing the source’s UUID/GUID and the ***current*** time (**not** a prior event’s time). Please note that the RTP-based protocol transports (which follow) do not have the same session and stream management issues.

9.4.1 “Transaction Response” Schema Definition (XSD; “metaTransResponse.xsd”)

The below XML schema definition defines the document used in transaction session flow types to acknowledge metadata. Some of the fields are optional since they are going to be application specific, and, as such, be specified by specific Working Groups within PSIA. However, the only required fields are:

- “**transactionSource**”: UUID/GUID of the metadata instance’s origin source.
- Either “**transactionTime**” (XML formats) or “**transactionNTPTime**” (GMCH formats): All acknowledgements MUST bear the timestamp of the metadata/event instance being acknowledged. However, the timestamp format is dependent upon the metadata format. XML formatted session use “transactionTime” while GMCH based sessions use “transactionNTPTime”.
- “**transactionStatus**”: All consumers MUST indicate their session status, with respect to data flow back to the source.

All other fields are, in general, optional but subject to specific PSIA Working Group classification. The schema definition below contains additional details on each element.

```
<?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:include schemaLocation="
http://www.psialliance.org/schemas/system/1.0/psiaCommonTypes.xsd"/>

<xs:element name="MetaTransactionResponse">
       <xs:complexType>
              <xs:sequence>
                     <xs:element name="transactionSource" minOccurs="1" maxOccurs="1"
                            type="GlobalID">
                        <xs:annotation>
                            <xs:documentation xml:lang="en">
                                   One of the following time types MUST be provided.
                                   For GMCH formatted data, the transaction ACK MUST
                                   pass back the NTP time of the original
                                   metadata/event instance. For XML
                                   formatted data, the original 'dateTime' MUST be
                                   passed  back.
                            </xs:documentation>
                        </xs:annotation>
                     </xs:element>
                     <xs:element name="transactionNTPTime" minOccurs="0" maxOccurs="1"
                            type="xs:hexBinary"/>
                     <xs:element name="transactionTime" minOccurs="0" maxOccurs="1"
                            type="xs:dateTime"/>
                     <xs:element name="transactionStatus" minOccurs="1" maxOccurs="1"
                            type="TransStatus"/>
                     <xs:element name="transactionID" minOccurs="0" maxOccurs="1"
                            type="xs:string">
                        <xs:annotation>
                            <xs:documentation xml:lang="en">
                                   If a transaction ID was in the MIDS of a
                                   metadata/event
                                   instance that is being ACK'd, that ID field MUST
                                   be passed back to the source.
                            </xs:documentation>
                        </xs:annotation>
                     </xs:element>
                     <xs:element name="transactionMIDS" minOccurs="0" maxOccurs="1"
                            type="xs:anyURI">
                        <xs:annotation>
                            <xs:documentation xml:lang="en">
                                   Returning the original instance's MIDS is desirable,
                                   but not required by the PSIA. However, certain WGs
                                   may require it for
                                   their specific types of transactions.
                            </xs:documentation>
                        </xs:annotation>
                     </xs:element>
                     <xs:element name="transactionDescr" minOccurs="0" maxOccurs="1"
                            type="xs:string"/>
                     <xs:element name="extTransHeader" minOccurs="0" maxOccurs="1"
                            type="TransExtension"/>
              </xs:sequence>
       </xs:complexType>
</xs:element>

<xs:simpleType name="TransStatus">
    <xs:annotation>
       <xs:documentation xml:lang="en">
              The following generic status strings have the following meanings:
              OK = transaction was received OK and the info was OK
              OK-MORE = The same as above but consumer wants next/more data (i.e.
              continue)
              OK-PAUSE = The same as above but consumer wants next/more data (i.e.
              continue)
              NOK = prior info was received but the contents/value were not OK/rejected
              NOK-MORE = prior data was received, contents/values were not OK, continue
              with more data.
              OK-PAUSE/NOK-PAUSE = The consumer indicates to the source the status of
              the contents/values of the prior received info, and to pause sending data.
```

```
                Any metadata/event occurrences that occur during a PAUSE are assumed to be
                lost. A pause is in effect until an OK/NOK-MORE is received. If a source
                cannot pause it returns an "HTTP/1.x 406 Not Allowed".
                Please note that the "OK" and "NOK" string values indicate that the
                consumer is not planning on proceeding with the session/datastream unless
                issued with an HTTP "GET" (this equals OK-MORE); However, "OK-MORE" and
                "NOK-MORE" can only
                be issued with "GET"s or HTTP/REST reponses, whereas the OK/NOKs can be
                issued  with DELETEs if the consumer is ready to close an active session
                (i.e. signal in advance that the consumer wants the session closed).
                It is required for ALL "NOK..." statuses, that the consumer also add a
                description ("transactionDescr") to provided additional information on the
                particular negative status condition. PSIA WGs MUST document their
                descriptive texts.
        </xs:documentation>
    </xs:annotation>
        <xs:restriction base="xs:string">
                <xs:enumeration value="OK"/>
                <xs:enumeration value="OK-MORE"/>
                <xs:enumeration value="OK-PAUSE"/>
                <xs:enumeration value="NOK"/>
                <xs:enumeration value="NOK-MORE"/>
                <xs:enumeration value="NOK-PAUSE"/>
        </xs:restriction>
</xs:simpleType>

<xs:complexType name="TransExtension">
        <xs:sequence>
                <xs:annotation>
                        <xs:documentation xml:lang="en">
                                The following element MUST be a unique string that
                                identifies the ordaining
                                body/group that has defined the header extension(s) and the
                                format thereof.
                                The ordaining body must publish/register its header
                                extensions with the PSIA
                                on it public forum, or in its external document forum. THe
                                format of the
                                string is a URI where the first field should identify the
                                group that defines
                                the header extension; in many cases this is either a PSIA
                                WG or a mfgr. The
                                following example is provided as a
                                guideline:"psia.SystemsWG/ExtraSourceIDs".
                        </xs:documentation>
                </xs:annotation>
                <xs:element name="ExtensionName" minOccurs="1" maxOccurs="1"
                        type="xs:anyURI"/>
                <xs:any namespace="##any" processContents="lax" minOccurs="0"
                        maxOccurs="unbounded" />
        </xs:sequence>
</xs:complexType>
</xs:schema>
```

The flow diagrams in Sections 9.2. and 9.3 give examples of the usage of the above schema in acknowledging transactional metadata. Please reference these diagrams for details.

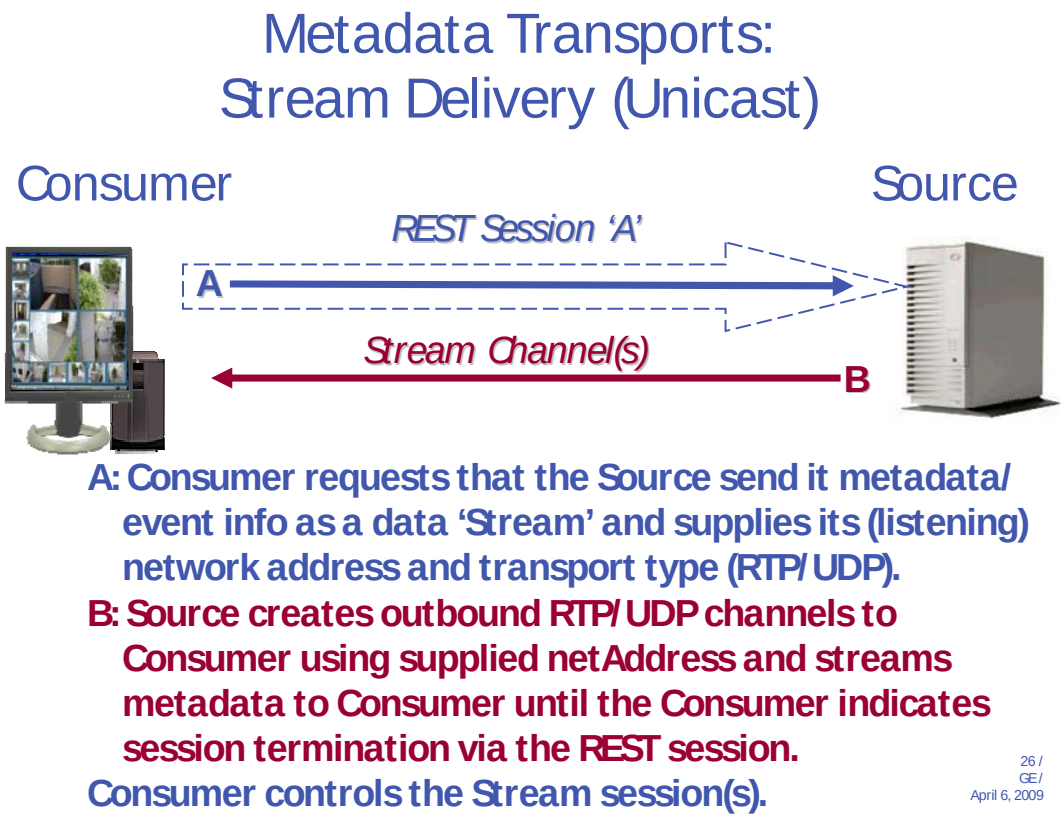
9.5 REST/RTP Streaming Transport Models ("RESTRTPStreamSrcOutUDP" and "RESTRTPStreamSrcOutTCP")

The next, remaining models are related to the 'Streaming' of metadata and event information. In a streaming environment, a REST session is used to setup the transport parameters, and RTP is utilized to transfer the metadata on a separate connection. The beauty of an RTP connection is based on the simplicity of its transport mechanics, which are much more

adept (i.e. higher efficiency, lower complexity) at moving information than HTTP related transports. Please note that the Group/Mass Notification model uses streaming in multicast mode as its metadata transport. Ergo, it is a subset of the overall streaming umbrella.

The basic Streaming model is aimed at systems where consumers desire low latency, and/or have connections that will have significant metadata traffic on them, and/or need to continue use of their REST management connection while the metadata is transferred. Also, the Streaming model requires less compute power than using an HTTP-based transport which makes it an ideal method for archive servers, etc. The primary drawback is that RTP-based transports, in general, are not as NAT/Firewall friendly as HTTP connections are. The following diagram depicts a basic streaming session.

Figure 9.5.1: Unicast Streaming (Overview)



Basically, in this scenario, a REST session is setup by the Consumer to convey the transport parameters for the metadata RTP connection(s). The Source uses the network address information to 'push' (i.e. send), via RTP, the metadata/event information to the network address designated by the Consumer. These streaming sessions are assumed to be somewhat long-lived in nature, but that is not a requirement. The following two diagrams depict basic Streaming model sessions over UDP (required) and TCP (optional) using unicast delivery.

Figure 9.5.2: Unicast-UDP Streaming (Message Flow Example)

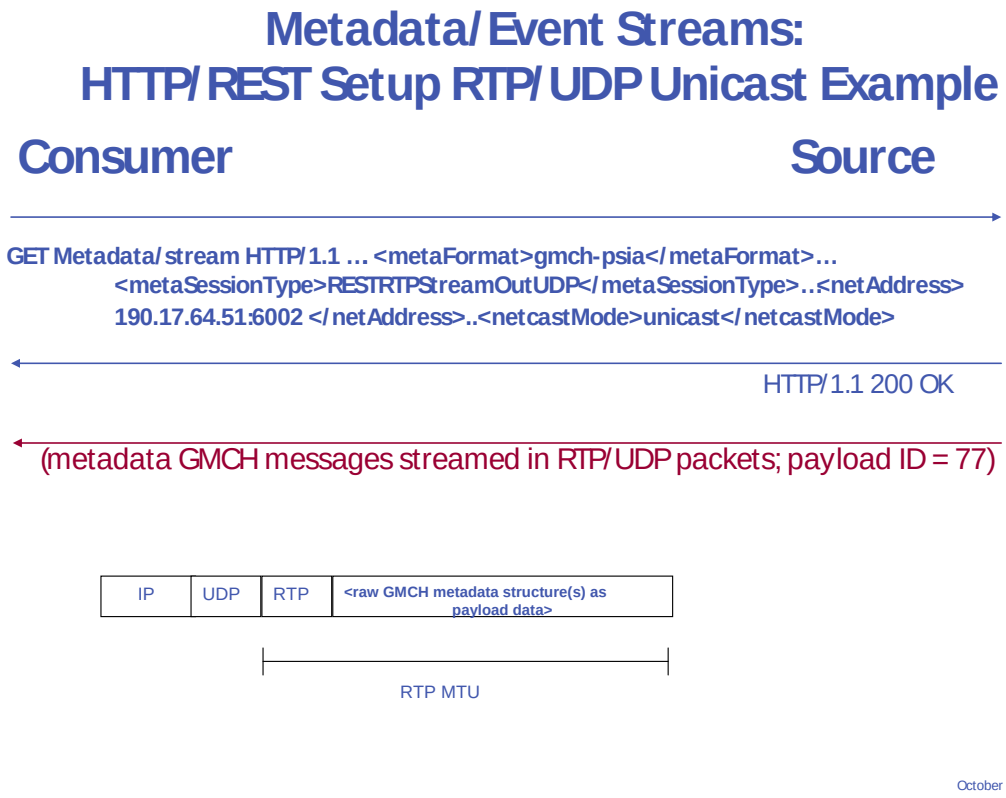
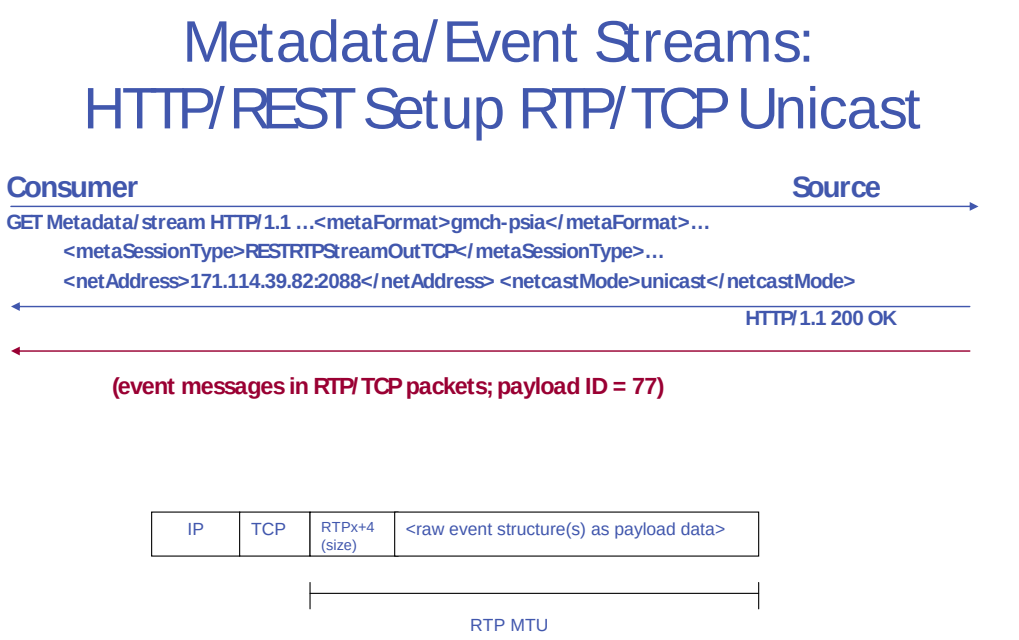


Figure 9.5.3: Unicast-TCP Streaming (Message Flow Example)



The Streaming models allow nodes to re-use RTP codebases already deployed for the support of audio/video transfer. And, in scenarios where audio and/or video are being played, with metadata, it allows a greater level of synchronization for the information streams involved. One minor technical note: in the Streaming models, including the Group/Mass Notification mode below, GMCH headers should not be split across MTUs when UDP transports are employed.

9.6 RTSP/RTP Streaming ("RTSPRTPStreamingSrcOut")

The prior section of this document covered REST-initiated streaming sessions using RTP as the media transport. This section describes the use of RTSP and SDP protocols as the session control protocols for RTP-based media streaming sessions. Please note that the transport characteristics and definitions for RTP/RTCP are the same as the REST/RTP sessions described above.

The use of RTSP (RFC 2326) and SDP (RFC 4566) as session control mechanisms enables the implementation of 'simple' client logic for consuming metadata/event data streams. In other words, clients do not need to 'read', or access, the other Metadata Service resources via REST to 'play' video and/or audio and/or metadata media streams. However, on exchange for the simplicity of using one set of protocols for streaming all forms of media, there is a tradeoff in the amount of control a consumer has regarding the types of information it receives. This is due to the protocol definitions for SDP, and due to the header and URI definitions, and practices, associated with RTSP. Given the desire to conform to industry standards, yet support a media

data type that is not currently defined for RTSP/SDP, this specification utilizes the standard definitions currently entailed by RFCs 2326 and 4566 and offers a set of optional, design compliant extensions that enable consumers to flexibly receive and manage metadata/event data streams using the RTSP/SDP control protocols.

First, a basic overview of RTSP/SDP is covered to set the background for the following definitions and descriptions. This overview does not contain certain details since the reader is directed to the pertinent RFCs to understand the details of each protocol.

RTSP (Real Time Streaming Protocol) is the protocol used as the control session for managing media streaming sessions. It is comprised of a set of session 'verbs' (i.e. methods), and header definitions, used to manage the state and content of RTP/RTCP streams. RTSP leverages the protocol work defined in the HTTP RFC 2068. SDP (Session Descriptor Protocol) contains the definitions for all the session and streaming related attributes. These attributes are advertised to consumers such that they can manage and consume data via RTSP(control)/RTP (media streams). So, SDP is the 'dictionary' of attributes that define a source's parameters to a consumer, and this information is conveyed to consumers via an RTSP session. In short, SDP is the session parameter dictionary, and RTSP is the communication 'pipe.' Via one RTSP/SDP session, multiple RTP/RTCP connections may be managed.

The basic command flow of an RTSP/SDP session consists of the following RTSP methods: DESCRIBE, SETUP, PLAY, TEARDOWN, and (sometimes) PAUSE. The order listed in the prior sentence is the usual order of occurrence, with the lone exception of PAUSE which may, or may not, be supported by a device. DESCRIBE is used to solicit the SDP descriptor from a source device using an RTSP URI to identify the context of the DESCRIBE (i.e. which resource is to be described). In order to maintain consistency, PSIA devices and systems MUST use the (RTSP-equivalent) PSIA REST resource URI to identify the specific resource(s) the consumer is asking the source to 'DESCRIBE.' For example, the URI "rtsp://PSIA/Streaming/channels/1" is the RTSP equivalent of the "/PSIA/Streaming/channels/1" IPMD-defined streaming resource for input channel # 1 (using RTSP notation). The use of the URI "/PSIA/Streaming/channels" would indicate that the consumer wants to see all of the streaming 'channels' offered by a source, though this may not be desirable for some device types. The reason for using the specific URI is to qualify the size of the SDP response. This is not as crucial for IPMD devices since they usually only have one, or two, video/audio channels. However, RaCM devices may have hundreds of live and archived channels/tracks, so qualifying/bounding the DESCRIBE request is a recommended practice for consumers. Please note that for RaCM compliant devices, the URI would start with "/PSIA/ContentMgmt/tracks…" for recorded content, "/PSIA/ContentMgmt/sources…" for source related descriptions (live and archived), and "/PSIA/ContentMgmt/channels…" for live input related descriptions.

Since consumers may ask for session description information in a general or specific fashion, certain rules apply to the content contained in a given SDP descriptor. Please note that any CMEM compliant device that supports RTSP/SDP/RTP session types, MUST always include metadata as a pseudo-'channel' in the SDP descriptor information under the following cases:

- The consumer issues a gratuitous (i.e. general) DESCRIBE where a specific channel ID is ***not*** present (e.g. "rtsp://PSIA/Streaming/channels"). In this case, metadata is always listed as a media stream in the response SDP since the consumer is wanting to get the attributes of all available streams.

- In cases where a channel ID is provided in a DESCRIBE, and there is metadata information that is related, or correlated, to that channel, metadata MUST be listed as an SDP media stream. This is an implicit reference by association. For example, assume that there is a dual-resolution IP Media device that supports Video Motion Detection (VMD). This device would have 2 ‘channels’, one for each video resolution. The video codec stream settings for each resolution would be accessible via the ‘/PSIA/Streaming/channels’ resource hierarchy. However, since VMD metadata is related to both channels, any DESCRIBE for channel 1 ***or*** 2 would need to have the metadata stream listed in the SDP response, also. This is due to the fact that the VMD metadata is related to both channels and is implicitly referenced. Please note that for RaCM devices, the correlation of metadata to other resources includes relationships to tracks and sources (i.e. input devices).

The above rules govern when and what metadata SDP information is provided to consumers.

9.6.1 SDP Usage In Metadata

Before going into the example message flow, below, it is significant to cover basic SDP structure and definitions. SDP defines the attributes related to streaming media sessions, codecs, and transports. RFC 4566 is the specification for the latest SDP standard. Readers are encouraged to read this spec to get a more detailed understanding of SDP. As noted above, consumers/clients issue an RTSP DESCRIBE request to get an SDP descriptor returned to them that defines the session and media attributes offered by a source. And, as noted above, URIs are used in RTSP DESCRIBE messages to identify the resources a consumer is interested in streaming; i.e. the DESCRIBE’s URI sets the resource ‘scope’ for getting session related attributes.

Given this information, the internals of SDP need to be covered next. In general, a specific instance of SDP information is called a ‘descriptor.’ A descriptor contains two primary types of information: A) Session information, and B) Media (i.e. stream related) information. Session information is global in nature whereas media information pertains to the individual attributes of particular media streams. Media stream information encompasses both transport protocol and basic codec properties. The following SDP descriptor is a basic example of information used by a hypothetical PSIA compliant, VGA resolution IP camera:

```
v=0
i=RTSP Session description
m=video 5000 RTP/AVP 96
i=H.264 AVC VGA video stream
a=control:rtsp://PSIA/Streaming/channels/1
a=rtpmap:96 H264/90000
a=fmtp:96 packetization-mode=1;profile-level-
id=4D400C;sprop-parameter-sets
=J01ADKkYUI/LgDUGAQa2wrXvfAQ=,KN4JF6A=a=
b=AS:1200
a=framerate:30
a=framesize:96 640-480
m=audio 5002 RTP/AVP 97
i=Audio stream for video
a=sendonly
```

```
a=control:rtsp://PSIA/Streaming/channels/1/audio
a=rtpmap:97 G726-32/8000
m=metadata 5004 RTP/AVP 98
i=PSIA Metadata Stream using GMCH format
a=control:rtsp//PSIA/Metadata/stream/gmch-psia
a=rtpmap:98 GMCH-PSIA/0
m=metadata 5004 RTP/AVP 99
i=PSIA Metadata Stream using XML format
a=control:rtsp//PSIA/Metadata/stream/xml-psia
a=rtpmap:99 XML-PSIA/0
...
```

The above SDP descriptor represents the following attributes:

- An H.264 VGA resolution RTP media stream ("m=video …") on UDP port 5000 with an RTP payload ID of 96. This is a 30 fps, 1.2Mbps stream with a SETUP/PLAY/TEARDOWN (S/P/T) URI of "rtsp://PSIA/Streaming/channels/1" ("a=control:…").
- A 32Kbps G.726 RTP audio stream on UDP port 5002 with an RTP payload ID of 97. This is a simplex audio channel from the source ("a=sendonly"). The S/P/T URI for this media stream is "rtsp://PSIA/Streaming/channels/1/audio".
- A PSIA metadata stream ("m=metadata 5004 RTP/AVP 98), in GMCH format, on UDP port 5002 with an RTP payload ID of 98. The media type is "GMCH-PSIA/0" which indicates that it has no specific bit rate. The S/P/T URI for this media stream is "rtsp://PSIA/Metadata/stream/gmch-psia". However, the URI, as with the above URIs, may be any string that uniquely identifies which stream is being referenced. Basically, the "control" URI ("a=control:…") is the 'handle' for each stream. The convention exemplified here is used for consistency and readability. The source could have easily used "rtsp://PSIA/Metadata/stream/channels/1/gmch-psia" if that format contained more usable information for the source.
- A PSIA metadata stream ("m=metadata 5004 RTP/AVP 98), in XML format, also on UDP port 5004 with an RTP payload ID of 99. The media type is XML-PSIA/0" which indicates that it has no specific bit rate. This metadata stream shares the same UDP port number since the choice between a GMCH format, or the XML format, is an either/or choice – not a simultaneous choice. It is also legal to place this media stream on a different UDP port. The S/P/T URI for this stream is "rtsp://PSIA/Metadata/stream/xml-psia".

9.6.1.1 SDP Media Stream Format Rules

The prior SDP descriptor example identifies some of the following ***rules*** and ***guidelines*** regarding RTSP-managed RTP metadata streams.

- ALL metadata media streams MUST be identified in the 'media' attribute as "metadata". E.g., "m=**metadata** 5001 RTP/AVP 77".
- ALL metadata media streams MUST be either of the RTP type "GMCH-PSIA/0" or "XML-PSIA/0" when mapping their type to a dynamic RTP payload ID. E.g. "a=rtpmap:96 GMCH-PSIA/0". When no 'rtpmap' is provided, the source and consumer MUST use RTP payload ID 77 as the static payload ID for PSIA metadata/event streams; however this only works in single format scenarios (i.e. GMCH and XML

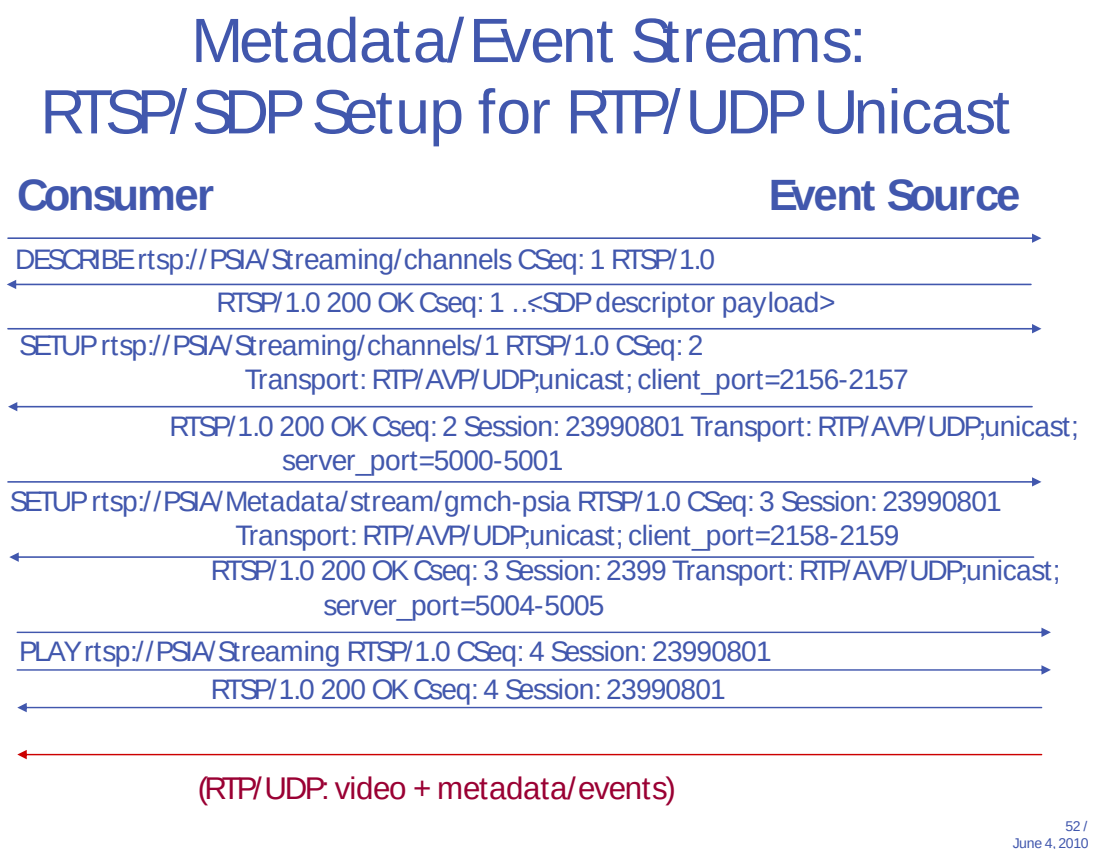
formatted streams CANNOT share the same payload ID). The use of dynamic RTP payload IDs is strongly recommended..

- An RTP control attribute URI ("a=control:…") MUST be provided such that the consumer and source can uniquely identify which media stream is being identified for SETUP/ PLAY/TEARDOWN operations. A source SHOULD structure the control URI with the prefix of "rtsp://PSIA/Metadata/stream...", for all metadata streams, such that there is consistency in identifying stream types. E.g. "rtsp://PSIA/Metadata/stream/channels/4/xml-psia " is applicable even though there is no direct correlation to a PSIA resource with that name.
- ALL video/audio streams SHOULD have a control attribute URI that correlates back to PSIA media source that is originating the data stream. E.g. "rtsp://PSIA/Streaming/channels/1" correlates to the "/PSIA/Streaming/channels/1" REST resource.

The above rules and guidelines are provided in order to aid in the consistency of implementation and thereby increase interoperability.

9.6.2 RTSP Usage for Metadata/Event Streams

The following message flow example depicts a consumer issuing a non-channel specific DESCRIBE.



The above message flow is based on the information revealed in the prior SDP example (see ***Section 9.6.1*** above). In this example RTSP session the consumer does the following:

- Requests SDP attributes for all the 'channels' on the Source by issuing a gratuitous DESCRIBE for the "rtsp://PSIA/Streaming/channel" resource which bears no channel ID value.
- The Source responds with the SDP descriptor information outlined in ***Section 9.6.1***.
- After processing the SDP descriptor information, the Consumer issues a SETUP for the video stream indicating that it will locally use the UDP ports of 2156 and 2157 for the RTP and RTCP connections. Please note that the Consumer uses the resource URI that was in the SDP descriptor for the video stream ("rtsp://Streaming/channels/1").
- The Source acknowledges the SETUP of the video channel by supplying a Session ID ("23990801"), since the nodes are now engaged in active session setup, and the Source indicates that its own source UDP port numbers for the RTP and RTCP sockets will be 5000 and 5001, respectively.
- Next, the Consumer issues a SETUP for the metadata/event stream using the control URI that was listed in the SDP descriptor ("rtsp://PSIA/Metadata/stream/gmch-psia"). The Consumer further indicates that it is locally creating UDP ports 2158 and 2159 for the RTP and RTCP connections, respectively.
- The Source then acknowledges the metadata/event stream SETUP and indicates that itsserver ports will be 5004 and 5005 for its end of the UDP RTP and RTCP connections.
- Next, the Consumer instructs the source to initiate the data streams by issuing a PLAY command with:
    - The original URI used in the DESCRIBE message (which is considered a 'global' resource URI), and…
    - the Session ID of 2399;

  the use of these fields in the PLAY command covers ***all*** of the streams that were already SETUP (video and metadata). See Section 9.6.2.1 below for details on the rules and guidelines regarding RTSP session management.

The above example, and the explanation, cover all the basic mechanics of managing video/audio and metadata/event streams. The only variations from the above mechanics occur when a consumer desires to specify session attributes, related to metadata, that are not currently described in the RTSP standard. These extensions are described below.

9.6.2.1 RTSP Rules and Extensions

RTSP is a well deployed session management protocol. Additionally, RTSP was designed as an extensible protocol so that it could be used for various media streaming applications. Its original design purpose was to address the management of audio/video media streams though provisions were made for other forms of data. Over the years various conventions have occurred in the use of RTSP that have stayed within the 'spirit' of RFC 2326 while not being explicitly being defined in the RFC. Since RTSP is very general in its specification, and most of the implementations used in the various industries have formed their own conventions for interoperability, PSIA recommends the following rules and guidelines in order to aid interoperability.

- RTSP/SDP Sources/servers MUST support the used of RTSP session IDs (i.e. "Session:…") as described in RFC 2326. A new, unique session ID is only assigned once the first SETUP message is received and is conveyed back to the consumer in a successful RTSP response to the SETUP message.
- RTSP resource URIs MUST be structured and used in the following manner:
    - The RTSP resource URI used on a DESCRIBE is the 'session global' URI. In other words, the DESCRIBE URI spans all of the media stream info, including the 'control' URIs for each potential stream, returned in an SDP descriptor.
    - Control URIs, i.e. those specified for each individual media stream within their respective sections of an SDP descriptor ("a:control:…"), are for use on managing individual streams; not sessions.
- All Sources MUST support the ability to perform session-level, *and* stream-level, PLAY and teardown commands. This operates in the following manner:
    - If an RTSP PLAY command's URI matches the original DESCIBE's resource URI, then this is a 'play all' command that is formed to play *all* of the prior setup streams. This is a session-level play.
    - If an RTSP PLAY command contains a resource URI that references a specific media stream, then the PLAY command is for a particular stream and only they stream should be activated irrespective of prior SETUP message sequences.
- All PAUSE and TEARDOWN messages follow the above rules applied to PLAY. A session level resource URI in a PAUSE or TEARDOWN message affect the entire session (i.e. all active streams). A command with a stream-specific resource URI only affects the specified media stream.
- All PSIA devices SHOULD support the RTSP PAUSE command. All sources MUST inspect the resource URIs in PAUSE and PLAY commands to determine which session or streams are affected per the above rules.
- Session level commands trump stream level commands, with the exception of SETUP. In other words, if a stream level message has been issued, the a session-level PAUSE, TEARDOWN, or a post-PAUSE PLAY occurs, the session level command affects all streams irrespective of prior state. For example, if a consumer PAUSE'd a single stream in a multi-stream session, and then issued a session-level TEARDOWN, all streams would be deactivated and the session prepared for termination.

In addition to the above rules, PSIA extends RTSP with a set of RTSP headers that allow a consumer to qualify the types of metadata/events that it desires to receive in a given RTP-based metadata/event stream. These new headers are:

- "**channels**": This RTSP header extension allows a consumer to designate that it only wants metadata/event information related to specific set of one, or more, channels. If more than one channel is specified, the channel IDs are separated by commas (i.e. comma separated variable (CSV) format). There may be ONLY one 'channels' header per SETUP message.
- "**category**": This header field allows consumers to provide MIDS information such that only metadata/event information specified by the supplied MIDS ('metaID') is to be sent. The format of the MIDS is the same domain/class/type… format specified in ***Section 7.2*** of this document. Please note that MIDS shorthand notation may be used (e.g. "//VideoMotion" indicates that the consumer wants all Video Motion occurrences

irrespective of domain or type contexts). There may be multiple ‘category’ headers per SETUP message.

- “**output**”: For those PSIA specifications that provide ‘profile’ levels for specifying the sets of function that comply with specific levels of their protocol definitions, this header allows the consumer to specify the profile ID/level that determines output content. For example, Video Analytics has ‘basic’ and ‘full’ output profiles. A consumer may specify the level of content output they desire using this extension header. The ‘output’ header MUST directly succeed the category affected by its presence. There may be multiple ‘output’ headers in a SETUP message but they MUST succeed the category they affect unless the category is implied by a channel designator, or is the only output offered.

The following RTSP message is provided to aid in understanding the usage of the above headers. Please note that these headers are germane only to RTSP SETUP messages since they are media stream type specific.

```
;;;;
SETUP rtsp://PSIA/Metadata/stream/gmch-psia RTSP/1.0
CSeq:3
Session:499A203BF7
channels:1,2
category://VideoMotion
category://VideoAnalytics/Alert
output:full
…
```

In the above example, the consumer issues a SETUP command for a metadata/event stream. In addition to the standard header fields “CSeq” and “Session”, the consumer specifies that it only wants metadata/event information related to input channels #1 and #2 (“channels:1,2”), and that it only wants VideoMotion metadata, and VideoAnalytics Alert metadata. Each instance of the ‘category’ header uses shorthand notation for the MIDS; i.e., the domain is not specified which is the equivalent of a ‘wild card’ (i.e. any, or all). Finally, PSIA’s Video Analytics standard specifies differing content output levels via the use of ‘profiles’. The “output” header has been added to the SETUP message to designate the output level of Video Analytics metadata information it desires to receive. Please note that the above example has multiple extension headers whereas there usually would not be

9.6.3 RTP Rules and Guidelines

The PSIA supported version of RTP is specified in RFC 3550, and the supported version of RTCP is described RFC 3551. The encapsulation of PSIA compliant CMEM data streams in RTP requires some additional definitions just as other audio/video data tyes do (e.g. RFC 3984, etc.). These additional qualfications are necessary due to the fact that all of the current RFCs governing the encapsulation and transport of media data are oriented towards rate-based audio/video data. Obviously, metadata/event streams are not rate-based in the same context as audio and video are. Additionally, metadata/event event information may be offered in GMCH encapsulated, or XML, formats. These formats affect certain aspects of RTP operation. The following RTP related rules and guidelines apply specifically to metadata/event streams.

- All PSIA nodes sending CMEM data in RTP MUST use the RTP ‘Marker’ bit (M-bit; RFC 3550, Section 5.1) in the following manner:

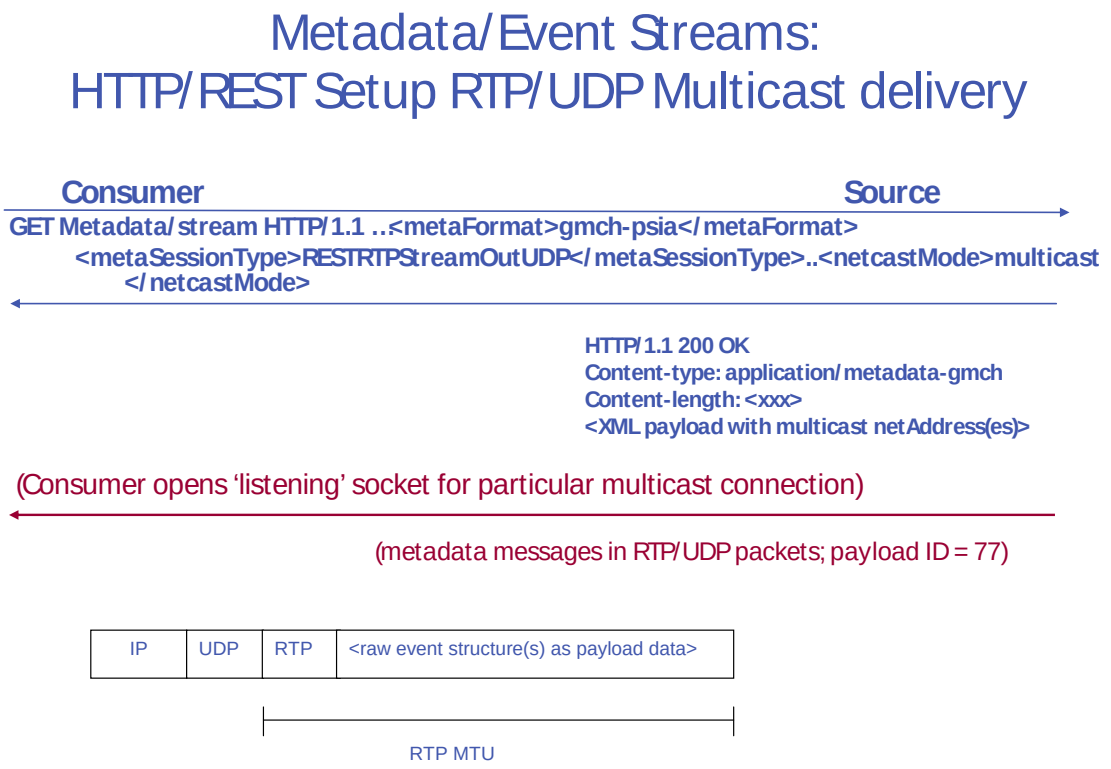
- **GMCH formatted payloads:** The Marker bit MUST be SET at the start of an RTP packet where the RTP payload, immediately succeeding the RTP header, starts with a GMCH encapsulation header. Since GMCH headers are self defining 'chunk' headers, a GMCH metadata/event instance may span multiple RTP packets, or an RTP packet may contain multiple GMCH instances, however, the only requirement is that GMCH instances that are aligned in an RTP packet are marked with a SET M-bit so that consumers may know which packets have ther GMCH information and which are 'continuation' packets (i.e. the M-bit is OFF).
- **XML formatted payloads:** The end/completion of each XML document instance MUST be marked with the M-bit SET in all RTP headers. If an XML document instance exceeds the length of an MTU and runs-over into another RTP packet, the M-bit would be OFF for all RTP packets that do not contain the final bytes of the respective XML document instance. Since RTP has no other boundary markers, there may be ONLY one XML document instance in an RTP packet that has the M-bit SET; i.e. there is ***no*** 'packing' of XML documents in a single RTP payload. .

- The RTP header 32-bit timestamp is a misnomer. In actuality this is not a timestamp (as in NTP or UTC structure), but a 'clock frequency' designator that is not coordinated to date/time or 'wall clock' time. Given this, this field is problematic for non-rate adapted data streams. In order to accommodate some well known implementation practices, PSIA specifies that this field is a monotonically increasing, unsigned 32-bit, payload instance counter. I.e the initial GMCH or XML object instance is assigned a random 32-bit unsigned integer value as the initial instance counter; each subsequent object instance receives a monotonically increasing number. If a GMCH or XML instance spans more than one RTP packet, then all of the RTP packets that comprise that object instance would bear the same counter value (The RTP packet sequence numbers would change, however).
- All other RFC 3550 rules apply with respect to RTP operation..

9.7 Group/Mass Notification Model ("RESTRTPStreamSrcOutUDP" with Multicast IP)

The Group/Mass Notification (GMN) Model is derived from the above Streaming models. The GMN model is based on multicast transport characteristics. This mode is advantageous for scenarios where there are multiple subscribers/consumers to any given metadata/event source. Additionally, access to multicast information has low-latency characteristics that may be necessary to meet latency demands in some system deployments (e.g. emergency response). The inhibitors for multicast deployment are basically, its inability to cross most VPN and NAT/Firewall boundaries without administrative intervention. The following diagram depicts a basic GMN session.

Figure 9.7.1: Multicast Streaming (Message Flow Example)



In the GMN scenario, the Source broadcasts metadata on 'channels'. Each 'channel' is a multicast UDP connection associated with set of metadata/event types. The metadata/event taxonomy listed in ***Section 3.1*** easily enables these capabilities of associating all, or some classes metadata information with specific channels/connections. In most cases, a single channel that carries all metadata/event information is used. Each connection can be setup to be permanent, i.e. active all the time, or dynamic, i.e. activated by a management request. This transport type is primarily feasible for server type applications, i.e. archive servers, event management servers, etc., and enables great scalability with low latencies, in campus environments.

9.8 Session/Transport Model Protocol Summary

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

Table 9.8.1: 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.
Asynchronous Notification Model	Client REST/HTTP	Client REST/HTTP 'Push'	Enables greater scalability with 'on

			demand' connection usage. More complex to implement.
REST/Streaming (unicast)	Client REST/HTTP	RTP/UDP, RTP/TCP (opt.)	Lightweight transport for connections that will move notable amounts of metadata/events. Uses IETF standards for data transport.
Group/Mass Notification Model (Streaming Multicast)	Client REST/HTTP	RTP/UDP Multicast	Mass scalability and low latency. Not VPN/Firewall friendly.

Please note that other transport models are also feasible, such as RTSP/RTP and AMQP, using the methods and formats listed in this document. In the following section of this document, the schemas used to describe and setup the above session types are described in detail.

9.8.1 Session Authentication

As mentioned in prior ***Sections 9.2.1*** and ***9.3.1***, 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 both SSLv3 and TLS v1 (RFC 4346) as the security options for HTTPS sessions.

10 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.

10.1 /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.

10.1.1 /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
Requirement Level	- All -			
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>
```

10.1.2 /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
Requirement Level	- All -		

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>
```

10.2 /PSIA/Metadata Information Resource Objects

The resource objects in this section of the specification pertain to the objects that define the categories, protocols and operation of metadata and 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.

10.2.1 /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
Requirement Level	- All -			
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>	
Notes	In the following XSD and example descriptions, the reading of the 'MetadataList' schema is done first. This is followed by the 'MetadataUpdate' XSD and examples for performing metadata updates.			

'MetadataList' XSD (filename="metadataList.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="MetadataList">
       <xs:complexType>
              <xs:sequence>
                     <xs:element name="numOfEntries" minOccurs="1" maxOccurs="1" type="xs:unsignedInt"/>
                     <xs:element name="metadataDescrList" minOccurs="1" maxOccurs="1"
                            type="MetadataDescrList"/>
              </xs:sequence>
          <xs:attribute name="version" type="xs:string" use="required" />
       </xs:complexType>
</xs:element>

<xs:complexType name="MetadataDescrList">
       <xs:sequence>
              <xs:element name="metadataDescriptor" minOccurs="1" maxOccurs="unbounded"
                     type="MetadataDescriptor"/>
       </xs:sequence>
</xs:complexType>

<xs:complexType name="MetadataDescriptor">
       <xs:sequence>
              <xs:element name="metaID" minOccurs="1" maxOccurs="1" type="MetadataIDDescr">
                     <xs:annotation>
                            <xs:documentation>
                                   This field carries the Domain-Class
                                   definitions for each Domain-Class available or supported
                            </xs:documentation>
                     </xs:annotation>
              </xs:element>
              <xs:element name="metaTypeList" minOccurs="0" maxOccurs="unbounded"
                            type="MetadataTypeList">
```

```
                        <xs:annotation>
                                <xs:documentation> This an optional list for describing the 'Types',
                                        of metadata-events within the above Domain-Class definition.
                                        It only needs to be provided, if known (i.e. by a source);
                                        proxies/brokers are not required to advertise types. Please
                                        note that only the 'Type' string needs to be supplied since
                                        the 'metaID' field (above) already supplies the Domain/Class.
                                </xs:documentation>
                        </xs:annotation>
                </xs:element>
                <xs:element name="metaChannelList" minOccurs="1" maxOccurs="1" type="MetaChannelList">
                        <xs:annotation>
                                <xs:documentation>
                                        The metadata-event information can be correlated, to the input
                                        channels using this list. The inclusion of this information in
                                        this schema allows consumers to get all the information needed
                                        in a single 'grab' for getting metadata via content or channel
                                        based definitions. Please note that the Metadata Service uses
                                        the RaCM definition of 'channels' (i.e. they are live inputs
                                        that a consumer may subscribe to via some specified mechanism).
                                        For certain metadata categories that are not necessarily 'bound'
                                        to a specific channel (e.g. '/PSIA/System...'), channel zero
                                        ('0') is the NULL channel.
                                </xs:documentation>
                        </xs:annotation>
                </xs:element>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="MetadataTypeList">
        <xs:sequence>
                <xs:element name="metadataType" minOccurs="1" maxOccurs="unbounded"
                        type="MetadataTypeDescr"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="MetadataIDDescr">
        <xs:sequence>
                <xs:element name="metdataMIDS" minOccurs="1" maxOccurs="1" type="xs:anyURI"/>
                <xs:element name="metadataPriority" minOccurs="1" maxOccurs="1" type="xs:unsignedByte"/>
                <!-- New for CMEM v1.1 -->
                <xs:element name="metadataItemMode" minOccurs="0" maxOccurs="1" type="ParamModMode"/>
        </xs:sequence>
</xs:complexType>

<!-- New for CMEM v1.1 -->
<xs:complexType name="MetadataTypeDescr">
        <xs:sequence>
                <xs:element name="metdataType" minOccurs="1" maxOccurs="1" type="xs:string"/>
                <!-- The following are optonal for Types they inherit Class attributes -->
                <xs:element name="metadataPriority" minOccurs="0" maxOccurs="1" type="xs:unsignedByte"/>
                <xs:element name="metadataItemMode" minOccurs="0" maxOccurs="1" type="ParamModMode"/>
        </xs:sequence>
```

```
</xs:complexType>

<xs:complexType name="MetaChannelList">
	<xs:sequence>
		<xs:element name="metaChannel" minOccurs="1" maxOccurs="unbounded" type="xs:unsignedInt"/>
	</xs:sequence>
</xs:complexType>

<!-- New for CMEM v1.1 -->
<xs:simpleType name="ParamModMode">
	<xs:annotation>
		<xs:documentation xml:lang="en">
			The default mode for any metadata item is 'read-write'. Therefore
			the only time this element type is required is when a source
			wants to lock an item as 'read-only' which prevents changing any
			of its characteristics.
		</xs:documentation>
	</xs:annotation>
	<xs:restriction base="xs:string">
		<xs:enumeration value="read-write"/>
		<xs:enumeration value="read-only"/>
	</xs:restriction>
</xs:simpleType>

</xs:schema>
```

Examples

```
<MetadataList version="1.1">
	<numOfEntries>3</numOfEntries>
	<metadataDescrList>
		<metadataDescr>
			<metaID>
				<metadataMIDS>/psialliance.org/VideoMotion</metadataMIDS>
				<metaPriority>4</metaPriority>
			</metaID>
			<metadataTypeList>
				<metadataType>
					<metadataMIDS>motionStart</metadataMIDS>
				</metadataType>
				<metadataType>
					<metadataMIDS>motionStop</metadataMIDS>
				</metadataType>
				<metadataType>
					<metadataMIDS>motion</metadataMIDS>
				</metadataType>
			<metadataTypeList>
			<metaChannelList>
				<metaChannel>1</metaChannel>
			</metaChannelList>
		</metadataDescr>
		<metadataDescr>
			<metaID>
```

```
                        <metadataMIDS>/psialliance.org/Config</metadataMIDS>
                        <metaPriority>3</metaPriority>
                    </metaID>
                    <metadataTypeList>
                        <metadataType>
                            <metadataMIDS>update</metadataMIDS>
                            <metaPriority>3</metaPriority>
                        </metadataType>
                    <metadataTypeList>
            </metadataDescr>
            <metadataDescr>
                    <metaID>
                        <metadataMIDS>/psialliance.org/System</metadataMIDS>
                        <metaPriority>3</metaPriority>
                    </metaID>
                    <metadataTypeList>
                        <metadataType>
                            <metadataMIDS>boot</metadataMIDS>
                            <metaPriority>4</metaPriority>
                        </metadataType>
                        <metadataType>
                            <metadataMIDS>fault</metadataMIDS>
                            <metaPriority>3</metaPriority>
                        </metadataType>
                        <metadataType>
                            <metadataMIDS>tamperAlarm</metadataMIDS>
                            <metaPriority>5</metaPriority>
                            <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=”1.0” xmlns=”urn:psialliance-org”>
        <numOfEntries>5</numOfEntries>
        <metadataDescrList>
            <metadataDescr>
                    <metaID>
                        <metadataMIDS>/psialliance.org/VideoMotion</metadataMIDS>
                        <metaPriority>4</metaPriority>
                    </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>
                        <metaPriority>3</metaPriority>
                </metaID>
                <metadataTypeList>
                        <metadataType>
                                <metadataMIDS>signalActive</metadataMIDS>
                        </metadataType>
                        <metadataType>
                                <metadataMIDS>signalInactive</metadataMIDS>
                                <metaPriority>4</metaPriority>
                        </metadataType>
                        <metadataType>
                                <metadataMIDS>signalError</metadataMIDS>
                                <metaPriority>4</metaPriority>
                        </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>
                        <metaPriority>3</metaPriority>
                </metaID>
                <metadataTypeList>
                        <metadataType>
                                <metadataMIDS>update</metadataMIDS>
                        </metadataType>
                <metadataTypeList>
        </metadataDescr>
        <metadataDescr>
                <metaID>
                        <metadataMIDS>/psialliance.org/System</metadataMIDS>
                        <metaPriority>3</metaPriority>
```

```
                        </metaID>
                        <metadataTypeList>
                                <metadataType>
                                        <metadataMIDS>boot</metadataMIDS>
                                </metadataType>
                                <metadataType>
                                        <metadataMIDS>fault</metadataMIDS>
                                </metadataType>
                        <metadataTypeList>
                </metadataDescr>
                <metadataDescr>
                        <metaID>
                                <metadataMIDS>/psialliance.org/Storage</metadataMIDS>
                                <metaPriority>3</metaPriority>
                        </metaID>
                        <metadataTypeList>
                                <metadataType>
                                        <metadataMIDS>mediaError</metadataMIDS>
                                </metadataType>
                                <metadataType>
                                        <metadataMIDS>mediaFailure</metadataMIDS>
                                </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’ XSD (metaUpdate.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:include schemaLocation="http://www.psialliance.org/schemas/system/1.0/metadataList.xsd"/>

<xs:element name="MetadataUpdate">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="numOfUpdates" minOccurs="1" maxOccurs="1" type="xs:unsignedInt"/>
                        <xs:element name="metadataIDDescr" minOccurs="1" maxOccurs="unbounded"
                                type="MetadataIDDescr"/>
                </xs:sequence>
            <xs:attribute name="version" type="xs:string" use="required" />
        </xs:complexType>
</xs:element>
</xs:schema>
```

Example

```
<MetadataUpdate version="1.1">
        <numOfEntries>2</numOfEntries>
        <metaDescr>
                <metaID>
                        <metadataMIDS>/metatdata.psia.org/Config/updzte</metadataNIDS>
                        <metaPriority>2</metaPriority>
                </metaID>
        </metaDescr>
        <metadataDescr>
                <metaID>
                        <metadataMIDS>/psialliance.org/System</metadataMIDS>
                        <metaPriority>2</metaPriority>
                </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 metadta 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 MatadataList 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" (type="metadataIDDescr")	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 advertised 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.

10.2.1.1 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.

10.2.2 /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
Requirement Level	- All -			
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				

Session Support XSD (filename="metaSessionSupport.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="1.1">

<xs:element name="MetaSessionSupport">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="metaFormats" minOccurs="1" maxOccurs="1"
                                type="MetaFormatList"/>
                        <xs:element name="metaSessionTypes" minOccurs="1" maxOccurs="1"
                                type="SessionTypeList"/>
                        <xs:element name="multicastCapable" minOccurs="1" maxOccurs="1"
                                type="xs:boolean"/>
                        <!-- For CMEM v1.0 nodes the following MUST be 'False';reserved for v1.1 and
```

```
                        later -->
                        <xs:element name="scheduleCapable" minOccurs="1" maxOccurs="1"
                                type="xs:boolean"/>
                        <!-- CMEM v1.1 field(s) -->
                        <xs:element name="queryParmsSupported" minOccurs="0" maxOccurs="1"
                                type="xs:boolean"/>
                </xs:sequence>
            <xs:attribute name="version" type="xs:string" use="required" />
        </xs:complexType>
</xs:element>

<xs:complexType name="MetaFormatList">
        <xs:sequence>
                <xs:element name="metaFormat" minOccurs="1" maxOccurs="unbounded" type="MetaFormat"/>
        </xs:sequence>
</xs:complexType>

<xs:simpleType name="MetaFormat">
        <xs:annotation>
                <xs:documentation xml:lang="en">
                        The following fields define the accepted PSIA formats
                        for PSIA metadata/event information.
                </xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
                <xs:enumeration value="gmch-psia"/>
                <xs:enumeration value="xml-psia"/>
        </xs:restriction>
</xs:simpleType>

<xs:complexType name="SessionTypeList">
        <xs:sequence>
                <xs:element name="metaSessionType" minOccurs="1" maxOccurs="unbounded"
                        type="SessionType"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="SessionType">
        <xs:sequence>
                <xs:annotation>
                        <xs:documentation xml:lang="en">
                                Only the first of the following 3 elements is required to be present.
                                This is because the default flow type for sessions is 'datastream'. Also,
                                only Asynch HTTP/REST sessions need to indicate if session parameters
                                persist across reboots, or not.The default is NO.
                        </xs:documentation>
                </xs:annotation>
                <xs:element name="metaSessionProtocol" minOccurs="1" maxOccurs="1"
                        type="SessionProtocolType"/>
                <xs:element name="metaSessionFlowType" minOccurs="0" maxOccurs="1"
                        type="SessionFlowType"/>
                <xs:element name="metaSessionPersistent" minOccurs="0" maxOccurs="1" type="xs:boolean"/>
        </xs:sequence>
</xs:complexType>
```

```
<xs:simpleType name="SessionFlowType">
	<xs:restriction base="xs:string">
		<xs:enumeration value="datastream"/>
		<xs:enumeration value="transaction"/>
		<xs:enumeration value="streamOrTransaction"/>
	</xs:restriction>
</xs:simpleType>

<xs:simpleType name="SessionProtocolType">
	<xs:annotation>
		<xs:documentation>
			The following transport/session types are arranged with
			'sending' session types first. Followed by 'receiving'
			session types next. Please note that the sending/
			receiving notation is always with respect to the node
			or device itself. Therefore the original session
			initiator must read a node's session types and then
			instruct that node on what type of session will be
			used for either sending or receiving data.
		</xs:documentation>
	</xs:annotation>
	<xs:restriction base="xs:string">
		<!-- The following are data sending session types -->
		<xs:enumeration value="RESTSyncSessionTargetSend"/>
		<xs:enumeration value="RESTAsyncSessionBackSourceSend"/>
		<xs:enumeration value="RESTRTPStreamSrcOutUDP"/>
		<!-- The following are optional or are reserved for TBD -->
		<xs:enumeration value="RESTRTPStreamSrcOutTCP"/>
		<xs:enumeration value="RTSPRTPStreamSrcOut"/>
		<xs:enumeration value="RTSPRTPStreamSrcOutInterleaved"/>
		<!-- The following are data receiving session types -->
		<xs:enumeration value="RESTAsyncSessionBackForReceive"/>
		<xs:enumeration value="RESTRTPStreamInUDP"/>
		<xs:enumeration value="RESTRTPStreamInTCP"/>
		<!-- The following is for 'event driven transmission only! -->
		<xs:enumeration value="EmailNotification"/>
	</xs:restriction>
</xs:simpleType>

</xs:schema>
```

Example(s)

```
<MetaSessionSupport version="1.1">
	<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>
</MetaSessionSupport>
```

The above example is for a PSIA compliant basic device. It offers its metadata in either GMCH encapsulated format, or as XML documents, in streaming mode. Additionally, the device complies with the required network session types that it supports.

```
<MetaSessionSupport version="1.1">
        <metaFormats>
                <metaformat>gmch-psia</metaFormat>
        </metaFormats>
        <metaSessionTypes>
                <metaSessionType>
                        <metaSessionProtocol>RESTSyncSessionInTargetSend</metaSessionProtocol>
                        <metaSessionFlowType>datastream</metaSessionFlowType>
                </metaSessionType>
                <metaSessionType>
                        <metaSessionProtocol>RESTAsyncSessionOutSourceSend</metaSessionProtocol>
                        <metaSessionFlowType>datastream</metaSessionFlowType>
                </metaSessionType>
                <metaSessionType>
                        <metaSessionProtocol>RESTRTPStreamOutUDP</metaSessionProtocol>
                        <metaSessionFlowType>datastream</metaSessionFlowType>
                </metaSessionType>
        </metaSessionTypes>
</MetaSessionSupport>
```

This example is more representative of a proxy/broker type device (maybe a RaCM device). Since it has to multiplex/aggregrate metadata from several sources, and offers metadata in a streaming manner, it only offers metadata in GMCH format. The session types it supports are the requires REST-based session support plus RTP streaming.

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

Element	Requirement Level	Notes
"metaFormats" (and "metaFormatList":: "metaFormat")	Required	An element comprosed of a list of the formats available for consumption. The choices are: • **"gmch-psia"**: GMCH encapsulation and description of metadata/event information. This format is REQUIRED, • **"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. This format is OPTIONAL.
“metaSessionTypes”:: “sessionTypeList”:: “sessionType”:: “metaSessionProtocol”	Required	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”	Dependent	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”	Dependent	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”	Required/ Dependent	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”	Required/ Dependent	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"	Optional/ Dependent	This element indicates whether, or not, the source device supports the use 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. A subscriber reads this schema instance to determine compatibility with the publisher. The supported formats are relatively simple. ALL nodes are required to offer GMCH encapsulation of their metadata/event information. Plus, if a device has a metadata format other than simple XML, such as text, binary or mixed-object formats, it MUST use GMCH encapsulation for transferring this data. If a source utilizes simple XML as its format, it may offer this format in addition to GMCH tagging.
The PSIA ordained session types for transferring metadata/event information are advertised via the following string tags. ***Please note that the construct of each tag is based on the relationship of the session initiator versus the data sender.*** *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	Require-ment Level	Notes
"RESTSyncSessionOutTargetSend"	Required	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***.
"RESTAsyncSessionBackSourceSend"	Required	Initial session is setup by consumer to target node. The consumer instructs, via session parameters, target node to re-establish a second, subsequent connection back to the consumer for sending data back to the consumer. Asynch = the source, once establishing the callback connection, can send without requiring a 'GET' from the consumer. This session type is described in ***Section 9.3***. Please note that the asynchronous HTTP/REST session types are the only ones that *may* require session login information to be conveyed as part of session initiation. Please see ***Section 10.2.4*** for

		details.
“RESTRTPStreamSrcOutUDP”	Optional for Basic devices; Required for Full/ Advanced devices;	Consumer sets-up a session via an HTTP/REST connection. The target/source initiates an outbound stream to the consumer via an RTP/UDP stream. These streams may be unicast or multicast based on session parameters. This session type is described in ***Section 9.4***. PSIA RaCM devices are required to support this session type in unicast mode.
"RESTRTPStreamSrcOutTCP"	Optional	The same as the above session type except the target/source node sends the streamed data in an RTP/TCP reliable connection. This session type is described in ***Section 9.4***.
“RTSPRTPStreamSrcOut” (new v1.1)	Optional	This session type is purely RTSP/SDP/ RTP/RTCP based. All of the following RFCs specify the standards for RTP streaming: • RFCs 2326/4556: RTSP/SDP control & description. • RFCs 3550/3551: RTP/RTCP streaming transport. This session type is new, and additional to the REST/RTP session type. Please note that this session type covers both UDP and TCP sessions. The transport choices are advertised in a source node’s SDP information. This session protocol is described in Section 9.6.
"RTSPRTPStreamSrcOutInterleaved"	Optional	This session type is a pure RTSP/RTP stream where the RTP data is ‘tunnelled’ back to the session initiator in the RTSP control session using RFC 2326 compliant ‘interleaving.’ No REST connection is used to manage the session.
**** The next 3 session types are about data RECEVING connections ****	---	---
"RESTAsyncSessionBackTargetReceive"	Optional	REST session initiator desires the target node to subsequently connect-back to it, via an HTTP/REST session, such that the session initiator may send data to the target node.
"RESTRTPStreamInUDP"	Optional	Publisher/advertiser accepts inbound REST managed RTP/UDP sessions for receiving metadata.
"RESTRTPStreamInTCP"	Optional	Same as the above session type except the publisher accepts reliable REST managed

RTP/TCP sessions for receiving data.

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.

10.2.3 /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, when necessary. A secondary benefit is that this resource allow administrators to determine which ‘channels’ are internal or external, and discover their characteristics.

URI	/PSIA/Metadata/channels		**Type**	Resource
Requirement Level	- All -			
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.			

Metadata Channels XSD (filename=”metaChannels.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:include schemaLocation="http://www.psialliance.org/schemas/system/1.0/psiaCommonTypes.xsd"/>
<xs:include schemaLocation="http://www.psialliance.org/schemas/system/1.0/metaSessionSupport.xsd"/>

<xs:element name="MetaChannels">
       <xs:complexType>
              <xs:sequence>
                     <xs:element name="numOfChannels" minOccurs="1" maxOccurs="1"
                            type="xs:unsignedInt"/>
                     <xs:element name="metaChannelList" minOccurs="1" maxOccurs="1"
                            type="MetaChannelList"/>
              </xs:sequence>
          <xs:attribute name="version" type="xs:string" use="required" />
       </xs:complexType>
</xs:element>
```

```
<xs:complexType name="MetaChannelList">
	<xs:sequence>
		<xs:element name="metaChannelParms" minOccurs="1" maxOccurs="unbounded"
				type="metaChannelParms">
			<xs:annotation>
				<xs:documentation>
				  The only requirement for a channel list is that the input/inbound
				  channels
				  MUST precede any output broadcast channels. See below for more
				  details on channel attributes.
				</xs:documentation>
			</xs:annotation>
		</xs:element>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="metaChannelParms">
	<xs:sequence>
		<xs:element name="channelID" minOccurs="1" maxOccurs="1" type="LocalID"/>
		<xs:element name="channelType" minOccurs="1" maxOccurs="1" type="channelSrcType"/>
		<xs:element name="metaFormatList" minOccurs="1" maxOccurs="1" type="MetaFormatList"/>
		<!-- Optional information fall below this line -->
		<!-- If the above 'channelType' is 'remoteSource' then one, or both,of the following must
		be provided -->
		<xs:element name="channelSrcGUID" minOccurs="0" maxOccurs="1" type="GlobalID"/>
		<xs:element name="channelSrcINetAddress" minOccurs="0" maxOccurs="1" type="xs:string"/>
		<!-- If the channel type = 'activeBroadcast' then the Multicast IP address MUST be
		provided below -->
		<xs:element name="multicastAddress" minOccurs="0" maxOccurs="1" type="xs:string">
			<xs:annotation>
				<xs:documentation xml:lang="en">
					IPv4 or IPv6 address, either with, or without, a
					'suffixed" port number, of the multicast stream.
					IPv4 Examples are:
					'239.110.1.57', or '239.66.4.31:4402'.
					IPv6 Examples are:
					'ff38:8000:0008:0000:0260:97ff:fe40:efab' or
					'ff37:8000:0017:0001:1040:8038:fa96:660c:5000'.
				</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="channelDescription" minOccurs="0" maxOccurs="1" type="xs:string"/>
		<!-- The following are Optional! -->
		<xs:element name="transactionAck" minOccurs="0" maxOccurs="1" type="xs:boolean">
			<xs:annotation>
				<xs:documentation>
					Determines if the event source needs explicit
					acknowledgments for each marked transaction.
					The default is 'False' (No).
				</xs:documentation>
			</xs:annotation>
		</xs:element>
	</xs:sequence>
```

```
</xs:complexType>

<xs:simpleType name="channelSrcType">
	<xs:annotation>
		<xs:documentation>
`			The following tag-values have these meanings:
			'internal'= metdata is internally generated by a process or task;
			'local'= data from a (local) port is formatted internally (e.g.
			serial I/O); 'remote'= a remote device or system is the orign
			of the metadata and this data is being proxied by the advertising
			node; 'activeBroadcast' = a currently active outbound multicast
			session that a consumer could 'join'.
		</xs:documentation>
	</xs:annotation>
	<xs:restriction base="xs:string">
		<xs:enumeration value="internalSource"/>
		<xs:enumeration value="localSource"/>
		<xs:enumeration value="remoteSource"/>
		<!-- Outbound channel type follows -->
		<xs:enumeration value="activeBroadcast"/>
	</xs:restriction>
</xs:simpleType>
</xs:schema>
```

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= »1.0 » 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 hypotherical encoder device has 5 input ‘channels’ of metadata information. The
first channel (“1”) is the Video Motion Detectin 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. 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. 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	Require-ment Level	Notes
“metaChannelParms”:: “channelID”	Required	The ASCII unsigned integer that is the ‘handle’ for the respective channel.
“metaChannelParms”:: “channelType”	Required	An enumerated string that identifies the type of input source that ‘drives’ a channel. Choices are: • “internalSource”: and internal process generates the metdata. • “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”	Required	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”	Dependent	If the above ‘channelType’ = “remoteSource” then the Proxy device MUST list the GUID/UUID of that remote device.
“metaChannelParms”:: “channelSrcINetAddress”	Dependent	If the above ‘channelType’ = “remoteSource” the Proxy device MUST list the IP network address of that source device.
“metaChannelParms”:: “multicastAddress”	Dependent	If the ‘channelType’ = “activeBroadcast” the source/proxy MUST list the multicast address, and port number, that correlate to that broadcast channel/session.
“metaChannelParms”::	Optional	A user-friendly descriptive string that helps

“channelDescription”		describe the channel further.
“metaChannelParms”:: “transactionAck”	Dependent	Nodes that require that metadata/event occurrences be ACK’d at transaction level, MUST indicate that requirement in this field.

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.

10.2.4 /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 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
Requirement Level	- All -		
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 below descriptions for QSP details.	Without Stream ID: <MetaSessionParms> With Stream ID: -None-	<ResponseStatus> OR For GETs with Asynch Stream IDs: <MetaSessionParms>
PUT	N/A	<TBD>	<ResponseStatus w/error code>
POST	***Conditional:*** See below descriptions for QSP details.	<MetaSessionParms>	<ResponseStatus> OR For GETs with Asynch Stream IDs: <MetaSessionParms>

DELETE	None	None	Delete is only used in cases: A) A consumer wants to terminate the current synchronous session after setting parameters for non-synchronous session; or… B) A consumer wants to delete the session resource instance for an asynchronous HTTP/REST session. In this case, the original session ID MUST be supplied as part of the resource URI (e.g. DELETE /Metadata/stream/9).
	This Metadata resource is mainly the access point for establishing (‘GET’ing) a Metadata/Event stream.		

Session Parameters XSD (filename=”metaSessionParms.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="1.1">

<xs:include schemaLocation=” http://www.psialliance.org/schemas/system/1.0/psiaCommonTypes.xsd”/>
<xs:include schemaLocation="http://www.psialliance.org/schemas/system/1.0/metaSessionSupport.xsd"/>

<xs:element name="MetaSessionParms" type="MetaSessionParms"/>

<xs:complexType name="MetaSessionParms">
       <xs:sequence>
              <xs:annotation>
                     <xs:documentation xml:lang="en">
                            There MUST only be ONE instance of the following element
                            except for the case where the session initiator is describing
                            asynchronous sessions that have 'alternate' or 'concurrent'
                            nodes that need to be notified when sessions are initiated by
                            the source. Synchronous 'GET' sessions are only allowed to
                            have one set of session parameters per request.
                     </xs:documentation>
              </xs:annotation>
              <xs:element name="metaXportParms" minOccurs="1" maxOccurs="unbounded"
                     type="MetaXportParms"/>
       </xs:sequence>
    <xs:attribute name="version" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="MetaXportParms">
       <xs:sequence>
              <!-- Added for CMEM v1.1: embedded session ID; previously it was in ResponseStatus -->
              <xs:element name="metaSessionID" minOccurs="1" maxOccurs="1" type="LocalID"/>
              <xs:element name="metaFormat" minOccurs="1" maxOccurs="1" type="MetaFormat"/>
              <xs:element name="metaSessionType" minOccurs="1" maxOccurs="1"
                     type="SessionProtocolType"/>
              <xs:element name="metaSessionFlowType" minOccurs="1" maxOccurs="1"
```

```
                        type="SessionFlowType"/>
                <!-- The following parameter is new for CMEM v1.1 -->
                <xs:element name="metaSessionRole" minOccurs="0" maxOccurs="1" type="SessionTargetRole"/>
                <xs:element name="metaSessionLogin" minOccurs="0" maxOccurs="1" type="SessionLogin"/>
                <!-- Updated/Changed for CMEM v1.1: individual network address fiels for target nodes -->
                <xs:choice>
                        <xs:element name="targetHostName" minOccurs="0" maxOccurs="1" type="xs:string"/>
                        <xs:element name="targetIPAddress" minOccurs="0" maxOccurs="1" type="xs:string"/>
                        <xs:element name="targetIPv6Address" minOccurs="0" maxOccurs="1"
                                type="xs:string"/>
                </xs:choice>
                <xs:element name="netcastMode" minOccurs="0" maxOccurs="1" type="NetCastMode"/>
                <xs:element name="transactionAck" minOccurs="0" maxOccurs="1" type="xs:boolean">
                        <xs:annotation>
                                <xs:documentation>
                                        Determines if the event source needs explicit
                                        acknowledgments for each transaction marked with
                                        an XID field (see CMEM spec).
                                        The default is 'False' (No).
                                </xs:documentation>
                        </xs:annotation>
                </xs:element>
                <xs:element name="metadataNameList" minOccurs="0" maxOccurs="1" type="MetadataNameList">
                        <xs:annotation>
                                <xs:documentation> This, and the following element, are only used for
                                        setting up CMEM sessions. They are not used by sources for
                                        advertising the session types and formats supported for
                                        metadata info transfer. These optional elements are used as
                                        filters for receiving data from a source. A consumer can
                                        apply metadata MIDS filters and/or chammel (input source)
                                        filters against a session that it desires to setup.
                                </xs:documentation>
                        </xs:annotation>
                </xs:element>
                <xs:element name="metadataChannelList" minOccurs="0" maxOccurs="1"
                        type="MetadataChannelList"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="SessionLogin">
        <xs:sequence>
                <xs:element name="authMode" minOccurs="1" maxOccurs="1" type="AuthenticationMode"/>
                <xs:element name="userLogin" minOccurs="0" maxOccurs="1" type="LoginInfo"/>
                <xs:element name="credentialObject" minOccurs="0" maxOccurs="1" type="LoginCredentials"/>
        </xs:sequence>
</xs:complexType>

<xs:simpleType name="AuthenticationMode">
        <xs:annotation>
                <xs:documentation xml:lang="en">
                        The following definitions cover the well-known, recommended
                        authentication and security definitions for HTTP sessions.
                </xs:documentation>
```

```
        </xs:annotation>
        <xs:restriction base="xs:string">
                <xs:enumeration value="none"/>
                <xs:enumeration value="basic"/>
                <xs:enumeration value="digest"/>
                <xs:enumeration value="https-SSLv3"/>
                <xs:enumeration value="https-TLSv1"/>
                <xs:enumeration value="https-TLSv1.1"/>
                <xs:enumeration value="https-TLSv1.2"/>
                <xs:enumeration value="https-TLSv1.x"/>
                <xs:enumeration value="any"/>
                <xs:enumeration value="other"/>
        </xs:restriction>
</xs:simpleType>

<xs:complexType name="LoginInfo">
        <xs:sequence>
                <xs:annotation>
                        <xs:documentation xml:lang="en">
                                Unused passwords should be listed as NULL XML elements
                        </xs:documentation>
                </xs:annotation>
                <xs:element name="userName" minOccurs="1" maxOccurs="1" type="xs:string"/>
                <xs:element name="password" minOccurs="1" maxOccurs="1" type="xs:string"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="LoginCredentials">
        <xs:sequence>
                <xs:annotation>
                        <xs:documentation xml:lang="en">
                                The following 'credentialType' field MUST be present to indicate the
                                type/format/content of any provided credential information. Reserved
                                strings are: 'SAML' and 'X.509'.
                                The 'credentialDefinition' URI is optional, but should be used as a
                                reference if a non-standard credential format/type is employed.
                        </xs:documentation>
                </xs:annotation>
                <xs:element name="credentialType" minOccurs="1" maxOccurs="1" type="xs:string"/>
                <xs:element name="credentialDefinition" minOccurs="0" maxOccurs="1" type="xs:anyURI"/>
                <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>

<xs:simpleType name="NetCastMode">
        <xs:annotation>
                <xs:documentation xml:lang="en">
                        The following modes pertain to 'streaming'
                        transport types. REST/HTTP sessions are
                        always unicast. Therefore, it only required
                        to specify the netCastMode when setting-up
                        metadata/event streams.
                </xs:documentation>
```

```
        </xs:annotation>
        <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="unicast"/>
                <xs:enumeration value="multicast"/>
        </xs:restriction>
</xs:simpleType>

<xs:complexType name="MetadataNameList">
        <xs:sequence>
                <xs:annotation>
                        <xs:documentation xml:lang="en">
                                The PSIA '/domain/class/type' URI notation for the metadata/event
                                information to be received.
                        </xs:documentation>
                </xs:annotation>
                <xs:element name="metadataIDString" minOccurs="1" maxOccurs="unbounded"
                        type="xs:anyURI"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="MetadataChannelList">
        <xs:sequence>
                <xs:element name="metaChannel" minOccurs="1" maxOccurs="unbounded"
                        type="LocalID"/>
        </xs:sequence>
</xs:complexType>

<!-- NEW for CMEM v1.1 -->
<xs:simpleType name="SessionTargetRole">
        <xs:annotation>
                <xs:documentation xml:lang="en">
                        Asynchronous sessions initiated by the Source can specify
                        more than one target receiver. In these cases the session
                        setup MUST specify if the nodes are 'primary' (i.e. they are
                        all concurrent recipients) OR if the session list is comprised
                        of alternates should a connection to a primary target fail. These
                        designators are impertinent for all other session types.
                </xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
                <xs:enumeration value="primary"/>
                <xs:enumeration value="alternate"/>
        </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Example(s)

```
<MetaSessionParms version=”1.1”>
        <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="1.1">
        <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="1.1">
        <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>
        </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).

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="1.1">
        <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>
        </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, initiating the data transfer. 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	Require-ment Level	Notes
"metaXportParms"	Required	Root element that contains the session and format parameters
"metaXportParms":: "metaSessionID"	Required	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"::	Required	Parameter that indicates the format of the metadata

“metaFormat:		to be transferred. Choices are: “gmch-psia”: General Metadata Classification Header format (Section 8.3). This format is required to be supported by all nodes. Additionally, this is the only format that works in non-HTTP/REST (i.e. RTP) transport types. “xml-psia”: For those nodes that have XML defined metadata/event schemas, this is an alternate option for HTTP/REST transports.
“metaXportParms”:: “metaSessionType:	Required	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.
“metaXportParms”:: “metaSessionFlowType”	Dependent	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”	Dependent	**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”	Dependent/ Required when Login info is present	(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”:: “userLogin”:: (“userName” & “password”)	Dependent/ Optional	The user name and password information is specific to asynchronous HTTP/REST notification sessions. If the node 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’)	Optional	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”	Dependent (Asynchro-nous notification)	(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 initator 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"	Dependent (Asynchro-nous notification)	(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"	Dependent	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"	Dependent/ Optional	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"	Optional	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"	Optional	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":: "multicastCapable:	Required	Boolean field that indicates to consumers/subscribers whether the metadata/event source supports UDP multicast transports, or not.
"metaXportParms":: "scheduleCapable"	Reserved	Required Boolean field for future use. All Version 1.0/1.1 nodes MUST indicate 'FALSE'.
"metaXportParms":: "queryParms"	Optional	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.

10.2.4.1 Query String Parameters (new v1.1)

CMEM Version 1.1 compliant nodes have the ability to support Query String Parameters (QSPs) on HTTP/REST 'GET' 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 a "RESTSyncSessionTargetSend" (Simple Reliable Get) metadata/event stream on the same HTTP session; and…
- C) the consumer has no more than 5 of the QSPs listed in the parameter table outlined below.

In these cases, the consumer/client is allowed to supply the session parameters via QSPs. All other session setup instances MUST supply the "MetaSessionParms' via an accompanying XML document. The allowable QSPs are listed below.

Table 10.2.4.1 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 "gmch-psia", or 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 desire 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 consymer: 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 comsumer wants all VideoAnalytics (Class) Alerts (Type). • "//Config/update: Indicates that the comsumer wants to be notified of configuration update events. All configuration events woule be "//Config". The above notation assumes that a source is either a pure PSIA node, or that other potential domains are either synonomous or completely different.
"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.

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.
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 HTTP/1.1
```

10.2.5 /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 "MetaSessionSupprt" 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
Requirement Level	Optional. Only devices/system supporting multicast delivery of Metadata/Events must support this 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				

Metadata Broadcast Parameters XSD (filename=”metaBroadcasts.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:include schemaLocation="http://www.psialliance.org/schemas/system/1.0/psiaCommonTypes.xsd"/>

<xs:element name="MetaBroadcasts">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="numOfBroadcasts" minOccurs="1" maxOccurs="1"
				type="xs:unsignedInt"/>
			<xs:element name="broadcastList" minOccurs="1" maxOccurs="1"
				type="broadcastList"/>
		</xs:sequence>
	   <xs:attribute name="version" type="xs:string" use="required" />
	</xs:complexType>
</xs:element>

<xs:complexType name="broadcastList">
	<xs:sequence>
		<xs:element name="broadcastSession" minOccurs="1" maxOccurs="unbounded"
type="BroadcastSession"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="BroadcastSession">
	<xs:sequence>
		<xs:element name="streamID" minOccurs="1" maxOccurs="1" type="xs:unsignedInt"/>
		<xs:element name="multicastAddress" minOccurs="1" maxOccurs="1" type="xs:string">
			<xs:annotation>
				<xs:documentation xml:lang="en">
					IPv4 or IPv6 address, either with, or without, a
					'suffixed" port number, of the multicast stream.
					IPv4 Examples are:
					'239.110.1.57', or '239.66.4.31:4402'.
					IPv6 Examples are:
					'ff38:8000:0008:0000:0260:97ff:fe40:efab' or
					'ff37:8000:0017:0001:1040:8038:fa96:660c:5000'.
				</xs:documentation>
			</xs:annotation>
		</xs:element>
		<xs:element name="metadataList" minOccurs="1" maxOccurs="1" type="metadataList"/>
		<xs:element name="channelList" minOccurs="0" maxOccurs="1" type="SrcChannelList"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="metadataList">
	<xs:sequence>
		<xs:element name="metaID" minOccurs="1" maxOccurs="unbounded" type="xs:anyURI"/>
	</xs:sequence>
</xs:complexType>
```

```
<xs:complexType name="SrcChannelList">
	<xs:sequence>
		<xs:element name="channelID" minOccurs="1" maxOccurs="unbounded" type="LocalID"/>
	</xs:sequence>
</xs:complexType>
</xs:schema>
```

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.

10.3 /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 catgories 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 supercedes, and replaces, the "/Custom/Event" service/resource hierarchy defined in Section 7.15 of the IPMD v1.1 specification.

10.3.1 /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
Requirement Level	Dependent; Required if the 'event' service is present.			
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>
```

10.3.2 /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
Requirement Level	Dependent; Required if the ‘event’ Service is present.			
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 devce 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
Requirement Level	Required		
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 mazimum 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 source 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 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 contraints prevent implementation.

10.3.4 /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
Requirement Level	Optional			
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.			

Event Schedule List XSD (filename=”eventScheduleList.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:include schemaLocation="eventSchedule.xsd"/>

<xs:element name="MetaEventScheduleList">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="eventSchedule" minOccurs="0" maxOccurs="unbounded"
                                type="MetaEventSchedule"/>
                </xs:sequence>
        <xs:attribute name="version" type="xs:string" use="required" />
    </xs:complexType>
</xs:element>
</xs:schema>
```

The above 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.

10.3.5 /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
Requirement Level	Optional/Conditional		
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.		

Event Schedule Item XSD (filename=”eventSchedule.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="MetaEventSchedule" type="MetaEventSchedule"/>

<xs:complexType name="MetaEventSchedule">
        <xs:sequence>
                <xs:element name="scheduleID" minOccurs="1" maxOccurs="1" type="xs:unsignedInt"/>
                <xs:element name="scheduleTimeRange" minOccurs="0" maxOccurs="1"
                                type="ScheduleTimeRange">
                        <xs:annotation>
                                <xs:documentation xml:lang="en">
                                        This optional field sets the Date/Time range of the
                                        following
                                        schedule time blocks. If this element is not present, it
                                        means
                                        that the subsequent schedule information is always active.
                                </xs:documentation>
                        </xs:annotation>
                </xs:element>
                <xs:element name="timeBlockList" minOccurs="1" maxOccurs="1"
                        type="TimeBlockList"/>
        </xs:sequence>
    <xs:attribute name="version" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="ScheduleTimeRange">
        <xs:sequence>
                <xs:element name="beginDateTime" minOccurs="1" maxOccurs="1" type="xs:dateTime"/>
                <xs:element name="endDateTime" minOccurs="1" maxOccurs="1" type="xs:dateTime"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="TimeBlockList">
        <xs:sequence>
                <xs:element name="timeBlock" minOccurs="1" maxOccurs="7" type="TimeBlock"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="TimeBlock">
        <xs:sequence>
                <xs:element name="dayOfWeek" minOccurs="1" maxOccurs="1" type="DayOfWeek"/>
                <!-- One of the 2 following time methods MUST be used per TimeBlock!! -->
                <xs:choice>
                        <xs:element name="timeSpanList" minOccurs="0" maxOccurs="1"
                                type="TimeSpanList"/>
                        <xs:element name="timeMapString" minOccurs="0" maxOccurs="1"
                                type="TimeMapString">
                            <xs:annotation>
                                <xs:documentation xml:lang="en">
                                        This is a 24 character field of boolean values (0 = false,
                                        1 = true) that denotes each hour of the day and indicates
                                        whether the event schedule is active for the corresponding
                                        hour in that day. The first character is 12:00AM. For
                                        example
                                        if someone wanted the schedule to be active from 7PM until
                                        5AM the pattern would be:"111111000000000000000111111".
                                </xs:documentation>
                            </xs:annotation>
                        </xs:element>
                </xs:choice>
        </xs:sequence>
</xs:complexType>

<xs:simpleType name="DayOfWeek">
        <xs:restriction base="xs:string">
                <xs:annotation>
                    <xs:documentation xml:lang="en">
```

```
                        Sunday = 1, Monday=2, Tuesday=3, etc...
                    </xs:documentation>
                </xs:annotation>
                <xs:enumeration value="1"/>
                <xs:enumeration value="2"/>
                <xs:enumeration value="3"/>
                <xs:enumeration value="4"/>
                <xs:enumeration value="5"/>
                <xs:enumeration value="6"/>
                <xs:enumeration value="7"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="TimeMapString">
        <xs:restriction base="xs:string">
                <xs:pattern value="[0-1]{24}"/>
        </xs:restriction>
</xs:simpleType>

<xs:complexType name="TimeSpanList">
        <xs:sequence>
                <xs:element name="timeSpan" minOccurs="1" maxOccurs="unbounded" type="TimeSpan"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="TimeSpan">
        <xs:sequence>
                <xs:element name="beginTime" minOccurs="1" maxOccurs="1" type="xs:time"/>
                <xs:element name="endTime" minOccurs="1" maxOccurs="1" type="xs:time"/>
        </xs:sequence>
</xs:complexType>

</xs:schema>
```

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	**Required/ Optional/ Conditional /Choice**	**Description**
"MetaEventSchedule:: scheduleID"	Required	Source assigned, unique ID for each schedule instance.
"MetaEventSchedule:: scheduleTimeRange" .. "MetaEventSchedule:: scheduleTimeRange:: beginDateTime/endDateTime	Optional	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 instance is 'active'. After the "endDateTime" the schedule instance remains, but it has no effect.
"MetaEventSchedule:: timeBlockList"	Required	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”	Required	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”	Required/ Choice	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”	Required/ Choice	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.

10.3.5.1 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>
```

10.3.6 /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 accomplaihed 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
Requirement Level	Conditionally Required when the ‘Actions’ Service is present.			
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.		
	Event Trigger List XSD (filename=”eventTriggerList.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:include
schemaLocation="http://www.psialliance.org/schemas/system/1.0/eventTrigger.xsd"/>

<xs:element name="MetaEventTriggerList">
    <xs:complexType>
        <xs:sequence>
                <xs:element name="eventTrigger" minOccurs="1" maxOccurs="unbounded"
                        type="EventTrigger" />
        </xs:sequence>
      <xs:attribute name="version" type="xs:string" use="required" />
    </xs:complexType>
</xs:element>
</xs:schema>
```

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 conditons 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.

10.3.7 /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…".
	Event Trigger XSD (filename="eventTrigger.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:include schemaLocation="http://www.psialliance.org/schemas/system/1.0/psiaCommonTypes.xsd"/>

<xs:element name="EventTrigger" type="EventTrigger"/>

<xs:complexType name="EventTrigger">
       <xs:sequence>
              <xs:element name="triggerID" minOccurs="1" maxOccurs="1" type="LocalID" />
              <xs:choice>
                     <xs:element name="eventCategories" minOccurs="0" maxOccurs="1"
                            type="EventCategoryList">
                        <xs:annotation>
                            <xs:documentation xml:lang="en">
                                   The (optional, if pertinent) Metadata/event categories that
                                   are
                                   supposed to 'trigger' the event processing
                            </xs:documentation>
                        </xs:annotation>
                     </xs:element>
                     <xs:element name="inputIOPortID" minOccurs="0" maxOccurs="1"
                            type="xs:string">
                        <xs:annotation>
                            <xs:documentation xml:lang="en">
                                   The (optional, if pertinent) port ID of the I/O port
                                   driving
                                   this source/trigger, if one exists.
                            </xs:documentation>
                        </xs:annotation>
                     </xs:element>
              </xs:choice>
              <xs:element name="eventTriggerNotificationList" minOccurs="1" maxOccurs="1"
                     type="EventTriggerNotificationList"/>
              <!-- Optional,qualifying items below -->
              <xs:element name="eventScheduleID" minOccurs="0" maxOccurs="1" type="LocalID" />
              <xs:element name="intervalBetweenEvents" minOccurs="1" maxOccurs="1"
                     type="xs:unsignedInt">
                   <xs:annotation>
                        <xs:documentation xml:lang="en">
                               Minimum interval, in seconds, between triggering events
                        </xs:documentation>
                   </xs:annotation>
              </xs:element>
       </xs:sequence>
    <xs:attribute name="version" type="xs:string" use="required" />
</xs:complexType>

<xs:complexType name="EventCategoryList">
       <xs:sequence>
              <xs:element name="eventCategory" minOccurs="1" maxOccurs="unbounded"
                     type="EventCategorySpec" />
       </xs:sequence>
</xs:complexType>

<xs:complexType name="EventCategorySpec">
       <xs:sequence>
              <xs:element name="eventMetaID" minOccurs="1" maxOccurs="1" type="xs:anyURI"/>
              <xs:element name="srcChannelID" minOccurs="0" maxOccurs="unbounded"
                     type="LocalID">
                  <xs:annotation>
                        <xs:documentation xml:lang="en">
                               If the device/system has metadata input channels,that are not part
                               of
                               a particular category (i.e. VideoMotion), then this is a potential
```

```
                              'filter' list for defining which channels are to be active input
                              for
                              the above category. The lack of a channel ID implies all channels
                              are
                              active.
                        </xs:documentation>
                    </xs:annotation>
                </xs:element>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="EventTriggerNotificationList">
        <xs:sequence>
                <xs:element name="eventTriggerNotificationMethod" minOccurs="1" maxOccurs="1"
                        type="LocalID" />
        </xs:sequence>
</xs:complexType>

</xs:schema>
```

The above 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	***Required/ Optional/ Conditional***	***Description***
"EventTrigger"	Required	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"	Required	Source assigned unique ID for the respective event trigger item.
Choice: "EventTrigger:: eventCategories" ***Or***… "EventTrigger":: inputIOPortId"	Required/ Choice	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"	Required	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"	Optional	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”	Optional	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… “EventTrigger:: “EventCategoryList:: EventCategorySpec:: srcChannelID”	Conditional (based on Choice)	**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 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 qualfiers 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. Please reference ***Section 10.3.5*** for more details on the scheduling of event trigger/notification operations.

10.3.8 /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
Requirement Level	Conditionally Required. Must be implemented if the 'Actions' Service is present.			
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:			
	Event Notification List XSD (filename="eventNotificationList.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:include
schemaLocation="http://www.psialliance.org/schemas/system/1.0/eventNotificationMethod.xsd
"/>

<xs:element name="EventNotificationList">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="eventNotification" minOccurs="1"
                                maxOccurs="unbounded" type="EventNotificationMethod"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
</xs:schema>
```

10.3.9 /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
Requirement Level	Conditionally Required. Must be implemented if the 'Actions' Service is present.			
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:			
	Event Notification XSD (filename="eventNotificationMethod.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:include schemaLocation="http://www.psialliance.org/schemas/system/1.0/psiaCommonTypes.xsd"/>
<xs:include schemaLocation="http://www.psialliance.org/schemas/system/1.0/metaSessionParms.xsd"/>

<xs:element name="EventNotificationMethod" type="EventNotificationMethod"/>

<xs:complexType name="EventNotificationMethod">
       <xs:sequence>
              <xs:element name="notificationMethodID" minOccurs="1" maxOccurs="1"
                     type="LocalID"/>
              <xs:choice>
                     <xs:annotation>
                         <xs:documentation xml:lang="en">
                             One, and only one, of the following Notification types MUST be
                             selected
                             per notification method instance.
                         </xs:documentation>
                     </xs:annotation>
                     <xs:element name="PSIASessionNotification" minOccurs="0" maxOccurs="1"
                            type="PSIANotificationParms"/>
                     <xs:element name="EmailNotification" minOccurs="0" maxOccurs="1"
                            type="EMailNotification"/>
                     <xs:element name="HTTPNotification" minOccurs="0" maxOccurs="1"
                            type="HttpNotification"/>
                     <xs:element name="FTPNotification" minOccurs="0" maxOccurs="1"
                            type="FtpNotification"/>
                     <xs:element name="IOSignaling" minOccurs="0" maxOccurs="1"
                     type="IOSignal"/>
              </xs:choice>
              <!-- The following qualify the above, but are mainly applicable to Email, HTTP,
              FTP and I/O -->
              <xs:element name="notificationRecurrence" minOccurs="1" maxOccurs="1"
                     type="NotificationRecur" />
```

```
                <xs:element name="notificationInterval" minOccurs="0" maxOccurs="1"
                        type="xs:unsignedInt">
                   <xs:annotation>
                        <xs:documentation xml:lang="en">
                                Minimum signal interval, in seconds, between event occurrence
                                notification.
                                This is ONLY needed when 'recur' is chosen as the notification
                                recurrence.
                        </xs:documentation>
                   </xs:annotation>
                </xs:element>
        </xs:sequence>
    <xs:attribute name=”version” type=”xs:string” use=”required”/>
</xs:complexType>


<!-- ======================================== -->
<!-- Common types                             -->
<!-- ======================================== -->

<xs:simpleType name="NotificationTypeName">
        <xs:restriction base="xs:string">
                <xs:enumeration value="PSIA"/>
                <xs:enumeration value="HTTP"/>
                <xs:enumeration value="Email"/>
                <xs:enumeration value="FTP"/>
        </xs:restriction>
</xs:simpleType>

<xs:complexType name="NetAddressType">
        <xs:sequence>
                <xs:annotation>
                        <xs:documentation xml:lang="en">
                                All sessions must choose if the target address for the
                                notification session is based on a host name (DNS),
                                an IPv4 address, or an IPv6 address.
                        </xs:documentation>
                </xs:annotation>
                <xs:choice>
                        <xs:element name="hostname" minOccurs="0" maxOccurs="1" type="xs:string"/>
                        <xs:element name="ipAddress" minOccurs="0" maxOccurs="1"
                                type="xs:string"/>
                        <xs:element name="ipv6Address" minOccurs="0" maxOccurs="1"
                                type="xs:string"/>
                </xs:choice>
                <!-- The following is optional for the IP address choices -->
                <xs:element name="portNumber" minOccurs="0" maxOccurs="1" type="xs:unsignedInt"/>
        </xs:sequence>
</xs:complexType>

<xs:simpleType name="NotificationRecur">
        <xs:restriction base="xs:string">
                <xs:enumeration value="beginning"/>
                <xs:enumeration value="end"/>
                <xs:enumeration value="recur"/>
        </xs:restriction>
</xs:simpleType>

<!-- The following are all related to Event Object Parameters -->

<xs:simpleType name="SnapshotFormatType">
        <xs:restriction base="xs:string">
                <xs:enumeration value="JPEG"/>
                <xs:enumeration value="GIF"/>
                <xs:enumeration value="BMP"/>
                <xs:enumeration value="PNG"/>
                <xs:enumeration value="TIFF"/>
                <xs:enumeration value="Private"/>
        </xs:restriction>
</xs:simpleType>
```

```
	<xs:simpleType name="VideoClipFormatType">
		<xs:restriction base="xs:string">
			<!-- The following is ISO/IEC 14496-14 -->
			<xs:enumeration value="MP4"/>
			<xs:enumeration value="AVI"/>
			<xs:enumeration value="ASF"/>
			<xs:enumeration value="WMV"/>
			<xs:enumeration value="MOV"/>
			<xs:enumeration value="M4V"/>
			<xs:enumeration value="Other"/>
		</xs:restriction>
	</xs:simpleType>

	<xs:complexType name="EventObjectParms">
		<xs:sequence>
			<xs:annotation>
				<xs:documentation xml:lang="en">
					An 'event object' can be an attachment, body, or file depending
					upon the notification type involved. Email needs body and/or
					attachment definition, where applied. Raw/plain HTTP needs body
					settings.  FTP needs the content of its file pushes defined.
					This type identifies parameters that sources need to advertise
					to consumers based on their capabililties.
				</xs:documentation>
			</xs:annotation>
			<xs:element name="metadataURIembedded" minOccurs="0" maxOccurs="1"
				type="xs:boolean"/>
			<xs:element name="metadataXMLembedded" minOccurs="0" maxOccurs="1"
				type="xs:boolean"/>
			<xs:element name="videoURIenabled" minOccurs="0" maxOccurs="1" type="xs:boolean"/>
			<xs:element name="videoSnapshotEnabled" minOccurs="0" maxOccurs="1"
				type="xs:boolean"/>
			<xs:element name="videoClipEnabled" minOccurs="0" maxOccurs="1" type="xs:boolean">
			    <xs:annotation>
				<xs:documentation xml:lang="en">
					If one of the above video options is chosen (snapshot or clip),
					then
					the corresponding choices MUST be selected by the user.
				</xs:documentation>
			    </xs:annotation>
			</xs:element>
			<xs:element name="snapshotFormat" minOccurs="0" maxOccurs="1"
				type="SnapshotFormatType"/>
			<xs:element name="videoClipFormat" minOccurs="0" maxOccurs="1"
				type="VideoClipFormatType">
			    <xs:annotation>
				<xs:documentation xml:lang="en">
					If 'video clips' are enabled, the following parameters need to be
					set if offered by the source. They are tenths of second units.
				</xs:documentation>
			    </xs:annotation>
			</xs:element>
			<xs:element name="preCaptureDuration" minOccurs="0" maxOccurs="1"
				type="xs:unsignedInt"/>
			<xs:element name="postCaptureDuration" minOccurs="0" maxOccurs="1"
				type="xs:unsignedInt"/>
		</xs:sequence>
	</xs:complexType>

	<!-- ======================================== -->
	<!-- PSIA Session notification parameters     -->
	<!-- ======================================== -->

	<xs:complexType name="PSIANotificationParms">
		<xs:sequence>
			<xs:element name="PSIASessionParms" minOccurs="0" maxOccurs="1"
				type="MetaSessionParms"/>
			<xs:element name="PSIASessionVideoSettings" minOccurs="0" maxOccurs="1"
				type="EventObjectParms"/>
		</xs:sequence>
	</xs:complexType>
```

```
<!-- ======================================== -->
<!-- The EMail notification parameter section -->
<!-- ======================================== -->

<xs:complexType name="EMailNotification">
	<xs:sequence>
		<xs:element name="receiverEmailAddress" minOccurs="1" maxOccurs="1"
			type="xs:string"/>
		<xs:element name="senderEmailAddress" minOccurs="1" maxOccurs="1"
			type="xs:string"/>
		<xs:element name="subjectLine" minOccurs="1" maxOccurs="1" type="xs:string"/>
		<xs:element name="bodySetting" minOccurs="0" maxOccurs="1"
			type="EventObjectParms"/>
		<xs:element name="mailAuthenticationMode" minOccurs="1" maxOccurs="1"
			type="MailAuthenticationMode"/>
		<!-- The following are dependent for Email accounts! -->
		<xs:element name="emailAccountName" minOccurs="0" maxOccurs="1" type="xs:string"/>
		<xs:element name="emailPassword" minOccurs="0" maxOccurs="1" type="xs:string"/>
		<xs:element name="networkAddress" minOccurs="0" maxOccurs="1"
			type="NetAddressType"/>
		<!-- All of the following elements are related to the POP EMail protocol or
		account -->
		<xs:choice>
			<xs:element name="popServerHostName" minOccurs="0" maxOccurs="1"
				type="xs:string"/>
			<xs:element name="popServerIPAddress" minOccurs="0" maxOccurs="1"
				type="xs:string"/>
			<xs:element name="popServerIPv6Address" minOccurs="0" maxOccurs="1"
				type="xs:string"/>
		</xs:choice>
	</xs:sequence>
</xs:complexType>

<xs:simpleType name="MailAuthenticationMode">
	<xs:restriction base="xs:string">
		<xs:enumeration value="none"/>
		<xs:enumeration value="SMTP"/>
		<xs:enumeration value="POP/SMTP"/>
	</xs:restriction>
</xs:simpleType>

<!-- ======================================== -->
<!-- The raw (non-PSIA) HTTP parameter section -->
<!-- ======================================== -->

<xs:complexType name="HttpNotification">
	<xs:sequence>
		<xs:element name="sessionType" minOccurs="1" maxOccurs="1"
			type="HttpSessionType"/>
		<xs:element name="networkAddress" minOccurs="1" maxOccurs="1"
			type="NetAddressType"/>
		<xs:element name="httpContents" minOccurs="1" maxOccurs="1"
			type="EventObjectParms"/>
		<!-- The following are optional for HTTP logins/accounts! -->
		<xs:element name="httpUserName" minOccurs="0" maxOccurs="1" type="xs:string"/>
		<xs:element name="httpPassword" minOccurs="0" maxOccurs="1" type="xs:string"/>
		<xs:element name="httpAuthenticationMethod" minOccurs="1" maxOccurs="1"
			type="HttpAuthMethods"/>
	</xs:sequence>
</xs:complexType>

<xs:simpleType name="HttpSessionType">
	<xs:restriction base="xs:string">
		<xs:enumeration value="http"/>
		<xs:enumeration value="https"/>
	</xs:restriction>
</xs:simpleType>

<xs:simpleType name="HttpAuthMethods">
	<xs:restriction base="xs:string">
```

```
                <xs:enumeration value="basic"/>
                <xs:enumeration value="MD5digest"/>
                <xs:enumeration value="none"/>
        </xs:restriction>
</xs:simpleType>


<!-- ==================================== -->
<!-- The FTP notification parameter section -->
<!-- ==================================== -->

<xs:complexType name="FtpNotification">
        <xs:sequence>
                <xs:element name="networkAddress" minOccurs="1" maxOccurs="1"
                        type="NetAddressType"/>
                <!-- The follwing are required for all FTP logins/accounts! -->
                <xs:element name="ftpUserName" minOccurs="1" maxOccurs="1" type="xs:string"/>
                <xs:element name="ftpPassword" minOccurs="1" maxOccurs="1" type="xs:string"/>
                <xs:element name="passiveModeEnabled" minOccurs="1" maxOccurs="1"
                        type="xs:boolean"/>
                <xs:element name="ftpUploadPath" minOccurs="1" maxOccurs="1" type="xs:string"/>
                <xs:element name="ftpBaseFileName" minOccurs="1" maxOccurs="1" type="xs:string"/>
                <xs:element name="ftpFileContents" minOccurs="1" maxOccurs="1"
                        type="EventObjectParms"/>
        </xs:sequence>
</xs:complexType>


<!-- ==================================== -->
<!-- The I/O Signaling parameters come next -->
<!-- ==================================== -->

<xs:complexType name="IOSignal">
    <xs:sequence>
        <xs:element name="outputIOPortID" minOccurs="1" maxOccurs="1" type="LocalID">
           <xs:annotation>
                <xs:documentation xml:lang="en">
                        The port ID of the I/O port to be activated
                </xs:documentation>
           </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

The above 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	**Required/ Optional/ Conditional /Choice**	**Description**
"EventNotificationMethod:: notificationMethodID"	Required	Source assigned, unique ID for each notification method instance.
		Users MUST Select 1 of the following 5 Options
"EventNotificationMethod::	Choice	This selection activates the use of a PSIA

”PSIASessionNotification”	(Required)	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”	Choice (Optional)	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”	Choice (Optional)	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”	Choice (Optional)	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”	Choice (Optional)	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”	Required	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/signling 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”	Conditional	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"	Required/ Choice	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.
"NetAddressType:: portNumber"	Optional	In some cases, a user may need to specify a particular port number for a session.
"EventObjectParms:: metadataURIembedded"	Optional/ Dependent	'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'
"EventObjectParms:: metadataXMLembedded"	Optional/ Dependent	'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).
"EventObjectParms:: videoURIenabled"	Optional/ Dependent	'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.
"EventObjecyParms:: videoSnapshotEnabled"	Optional/ Dependent	'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).
"EventObjectParms:: videoClipEnabled"	Optional/ Dependent	'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.
"EventObjectParms:: snapshotFormat"	Dependent	'snapshotFormat': If snapshots are supported, the source must supply them in a known format. *JPEG is the preferred format (see XSD).*

"EventObjectParms:: videoClipFormat"	Dependent	'videoClipFormat': If video clips are supported they must conform to one of the known formats (see XSD). *ISO 14496-14 (MP4) is the preferred format.*
"EventObjectParms:: preCaptureDuration/ postCaptureDuration"	Optional/ Dependent	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).

10.3.9.1 REST Aysnchronous 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 consunmer wants snapshots to accompany GMCH or XML event information. GMCH events will send video snapshots as additional objects with MOH headers. 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 consunmer wants video clips to accompany GMCH or XML event information. GMCH events will send video clips as additional objects with MOH headers. 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.

10.3.9.2 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	Required/ Optional/ Conditional /Choice	Description
"EmailNotification:: receiverEmailAddress"	Required	Email name/account of the target recipient.
"EmailNotification:: senderEmailAddress"	Required	Email name/account to be used by the sender for this notification instance.
"EmailNotification:: subjectLine"	Required	Subject line content to be listed for each Email notification.
"EmailNotification:: bodySetting"	Optional	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:: mailAuthenticationMode"	Required	Consumer must select one of the following options: • "**none**": No authentication is to be performed

		for the Email session. • "**SMTP**": SMTP authenication is to be used for the session. • "**POP/SMTP**": POP/SMTP authentication is to be used for the Email session.
"EmailNotification:: emailAccountName"	Optional/ Dependent	If the remote, target account requires a login, then the account/user name needs to be provided.
"EmailNotification:: emailPassword"	Optional/ Dependent	If the remote, target account requires a login, then the password needs to be provided if one is required..
"EmailNotification:: networkAddress"	Optional	If required the network address (IP address or host name) of the target Email recipient.
"EmailNotification:: popServerHostName / popServerIPAddress / popServerIPv6Address"	Optional/ Dependent (choice)	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.

10.3.9.3 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	**Required/ Optional/ Conditional /Choice**	**Description**
"HttpNotification:: sessionType"	Required	User MUST select either "**http**" or "**https**" as the session type.
"HttpNotification:: networkAddress"	Required (choice)	User MUST provide either the IPv4/IPv6 address, or host name, of the target HTTP recipient.
"HttpNotification:: bodySetting"	Required	User MUST select from a choice of paramters 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 paramters (see EventObjectParms in preceding table).
"HttpNotification:: httpUserName / (and) httpPassword"	Optional	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"	Required	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.

10.3.9.4 FTP Notification Method Parameters

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

Element Name	**Required/ Optional/ Conditional /Choice**	**Description**
"FtpNotification:: networkAddress"	Required (choice)	User MUST provide either the IPv4/IPv6 address, or host name, of the target FTP server.
"FtpNotification::	Required	User name of the FTP login required at the FTP

ftpUserName”		server.
“FtpNotification:: ftpPassword”	Required	Password of the FTP login required at the FTP server (may be empty; i.e. “”).
“FtpNotification:: passiveModeEnabled	Required	User MUST indicate if PASSIVE (true) or ACTIVE (false) FTP session mode is to be employed.
“FtpNotification:: ftpUploadPath”	Required	Path of where the Source is to deposit the event information.
“ftpNotification:: ftpBaseFileName”	Required	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”	Required	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 contentsbody 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

		paramters (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).

10.3.9.5 I/O Signaling Notification Method Paramters

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.

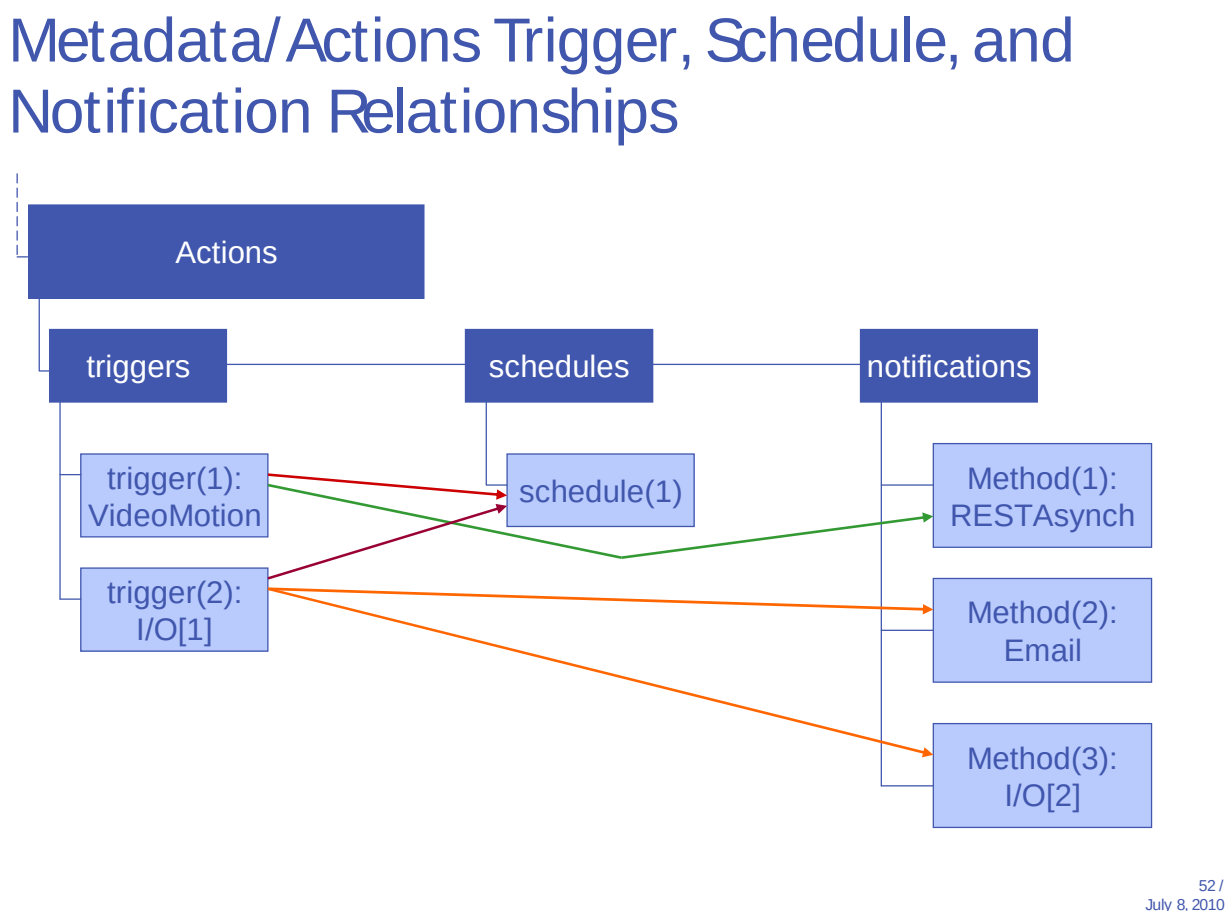
Element Name	**Required/ Optional/ Dependent**	**Description**
"IOsignal:: outputIOPortID"	Required	The ID of the output I/O channel ("/PSIA/ System/IO/<id>) to be activated when an event occurs.

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

The above subsections of Section 10.3 prescribe the data defintions, 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.

10.3.10.1 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 wil 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 refences 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”).

10.3.10.2 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 confirming 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 employee many options, may 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</dcheduleID>
        <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>
                                <timeSpam>
                                        <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</eventNotificationMethod>
            </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.

```
    OST /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</eventNotificationMethod>
                <eventTriggerNotificationMethod>3</eventNotificationMethod>
        </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.

11 How 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 protocos 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.

11.1 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.

11.2 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.

11.3 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.

11.4 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.

External Document References

[RFC 1305]	"Internet Time Synchronization: the Network Time Protocol", D. L. Mills, March 1992 URL:http://www.ietf.org/rfc/rfc1305
[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
[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 Descripton 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

Appendix A : PSIA Common Types XSD (psiaCommonTypes.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:psialliance-org" elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="urn:psialliance-org"
version="0.1">

<xs:simpleType name="GlobalID">
       <xs:annotation>
            <xs:documentation xml:lang="en">
               The representation of a GUID, generally the system(ic) id of an element.
       </xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:pattern
                value="\{[a-fA-F0-9]{8}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-fA-F0-9]{4}-[a-
fA-F0-9]{12}\}"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="LocalID">
       <xs:annotation>
            <xs:documentation xml:lang="en">
               The representation of a 'Local ID' is based on an unsigned
               integer which represents, basically, the index in a resource
               list for a particular item or object. The Local ID is to be used for
               channels, tracks, zones, areas, regions, hardware I/O ports,
               etc. Please note that 'zero' (0) is the NULL ID which indicates
               that a new element/resource needs to be allocated.
           </xs:documentation>
       </xs:annotation>
       <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>

<xs:complexType name="TimeSpan">
       <xs:sequence>
              <xs:element name="startTime" minOccurs="1" maxOccurs="1"
                     type="xs:dateTime"/>
              <xs:element name="endTime" minOccurs="1" maxOccurs="1"
                     type="xs:dateTime"/>
       </xs:sequence>
</xs:complexType>

<xs:simpleType name="ContentType">
       <xs:annotation>
              <xs:documentation xml:lang="en">
                     Types of content that can be searched or retrieved
              </xs:documentation>
       </xs:annotation>
       <xs:restriction base="xs:string">
              <xs:enumeration value="video"/>
              <xs:enumeration value="audio"/>
              <xs:enumeration value="metadata"/>
              <xs:enumeration value="text"/>
              <xs:enumeration value="mixed"/>
              <xs:enumeration value="other"/>
       </xs:restriction>
</xs:simpleType>

<xs:simpleType name="BaseSizeUnit">
       <xs:restriction base="xs:string">
              <xs:annotation>
                     <xs:documentation> The following tags cover storage units
                            in megabytes (MBs), gigabytes (GBs), terabytes (TBs),
                            petabytes (PBs), exabytes (XBs), mebibytes (MiBs),
                            and Gibibytes (GiBs).
```

```
                        </xs:documentation>
                </xs:annotation>
                <xs:enumeration value="MBs"/>
                <xs:enumeration value="GBs"/>
                <xs:enumeration value="TBs"/>
                <xs:enumeration value="PBs"/>
                <xs:enumeration value="XBs"/>
                <xs:enumeration value="MiBs"/>
                <xs:enumeration value="GiBs"/>
        </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Appendix 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, 0x6dde4eb, 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
```

Appendix C: GMCH H/Include File

```
/*-------------------------------------------------------------------------
 *
 * PSIA General Metadata Classification Header (GMCH) include file
 *
 * Copyright 2009, 2010 Physical Security Interoperability Alliance (PSIA)
 *
 * This source code is distributed as reference code for all PSIA members.
 * It is made availabe AS-IS with no expressed or implied warranties. This
 * file may not be distributed except internal to those companies that
 * are PSIA members and only for development purposes. This code is for
 * reference purposes only.
 *
 *-------------------------------------------------------------------------*/

/* Protect against multi-inclusion ===*/

#if !defined(__GMCH__)

#define __GMCH__ 1

/* Add support for C++ inclusion ====*/

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



/* Types, etc. ------------------------------------------------*/

#if !defined(_BASETSD_H_) /* Don't redefine MSVC types */

#if !defined(BYTE)
       #define BYTE unsigned char
#endif
#if !defined(UINT16)
       #define UINT16 unsigned short int
#endif
#if !defined(UINT32)
       #define UINT32 unsigned int
#endif
#if !defined(UINT64)
       #define UINT64 unsigned long long
#endif

#endif

/* Constants, etc. -------------------------------------------*/

#define GMCH_SIGNATURE '/GMCH\r\n\0'
#define GMCH_SIGNATURE_LEN 8
#define GMCH_MIME_TYPE “application/metadata-gmch”
#define GMCH_STREAM_MIME_TYPE “multipart/x-gmch-stream”
#define GMCH_VERSION1 0x0100

#define __GUID_LEN__ 16
#define U32GUIDLEN (__GUID_LEN__/sizeof(UINT32))
#define U64GUIDLEN (__GUID_LEN__/sizeof(UINT64))

/* Values for the GMCH type field ========*/

enum GMCH_TYPE{ GMCH_TYPE_SIMPLE_BIN=0, GMCH_TYPE_SIMPLE_XML,
       GMCH_TYPE_SIMPLE_TEXT, GMCH_TYPE_COMPLEX };


/* Structures, unions, etc. ----------------------------------------------*/
```

```
/* UUID/GUID overlay for various access methods ===*/

typedef union
{
       BYTE           guidStrg[__GUID_LEN__];
       UINT32         guidU32[U32GUIDLEN];
       UINT64         guidU64[U64GUIDLEN];
} _GUID_;


/* GMCH structure definition ============*/

typedef struct GMCH
{
       BYTE           gmchSignature[GMCH_SIGNATURE_LEN];
       UINT16         gmchVersion;
       BYTE           gmchPriority;
       BYTE           gmchType;
       UINT32         gmchSize;
       _GUID_         gmchSrcID;
       _GUID_         gmchLinkID;
       UINT64         gmchTime;
       UINT32         gmchSrcLocalID;
       UINT16         gmchNumOfObjects;
       UINT16         gmchMIDSLen;
       UINT32         gmchDomainBTag;
       UINT32         gmchClassBTag;
       UINT32         gmchTypeBTag;
       /* Metadata/Event MIDS (URI) goes here!) */
}  GMCH;

       /* This is the same as the above, it
          just includes the MIDS str field */
typedef struct _GMCH
{
       BYTE           gmchSignature[GMCH_SIGNATURE_LEN];
       UINT16         gmchVersion;
       BYTE           gmchPriority;
       BYTE           gmchType;
       UINT32         gmchSize;
       BYTE           gmchSrcID[__GUID_LEN__];
       BYTE           gmchLinkID[__GUID_LEN__];
       UINT64         gmchTime;
       UINT32         gmchSrcLocalID;
       UINT16         gmchNumOfObjects;
       UINT16         gmchMIDSLen;
       UINT32         gmchDomainBTag;
       UINT32         gmchClassBTag;
       UINT32         gmchTypeBTag;
       BYTE           midStr[];
}  _GMCH;

/* Multi-Object Header struct def'n =====*/

typedef struct MOH
{
       UINT32         objectSize;
       UINT16         mimeStrLen;
       /* MIME def'n string goes here! */
} MOH;

       /* this is the same as the above, it
          just includes the MIME str field */
typedef struct _MOH
{
       UINT32         objectSize;
       UINT16         mimeStrLen;
       BYTE           mimeTyprStr[];
} _MOH;
```

```
/* Assorted macros, etc. -----------------------------------------*/

       /* The following macro maps to the std 'strcmp' return values */
#define CMPGUIDSTR(pGuid1, pGuid2) \
       strncmp( (char *)pGuid1, (char *)pGuid2, __GUID_LEN__)

       /* The following macro maps to the std boolean true/false comparison logic */
#define CMPGUID(pGuid1, pGuid2) \
        ( ((( _GUID_ *pGuid1)->guidU64[0]) == ((_GUID_ *pGuid2)->guidU64[0])) && \
          ((( _GUID_ *pGuid1)->guidU64[1]) == ((_GUID_ *pGuid)2->guidU64[1]))  )


#if defined(__cplusplus)
}
#endif

#endif /* EndIF of multi-inclusion protection */
```

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.
- “**motion:Left**”: 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”.`

Appendix 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`”.

Appendix 3: “Config” Metadata Class Dictionary

Binary Tag (CRC32) = 0xD3262A4A

The following ‘type; fieds/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”.`

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”.`

Appendix 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 usuall set via a configurable threshold).

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

Appendix 6: “PointOfSale” Metadata Class Dictionary

Binary Tag (CRC32) = 0x9212C808

The following ‘type; fieds/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”.`

Appendix 7: “System” Metadata Class Dictionary

Binary Tag (CRC32) = 0xCEE114BD

The following ‘type; fieds/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`”.

Appendix 8: “Storage” Metadata Class Dictionary

Binary Tag (CRC32) = 0x9BC722A8

The following ‘type; fieds/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”.`

Appendix 9: “Metadata” Metadata Class Dictionary

Binary Tag (CRC32) = 0xB6625642

The following ‘type; fieds/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.

An example of one of the above is:
“/psialliance.org/Metadata/updatedCategories”.

Appendix 10: “Network” Class Dictionary

Binary Tag (CRC32) = 0x

The following ‘type; fieds/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.

- “**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 keepalive for HTTP metadata streaming sessions that have time gaps without information. This type does not have a payload; just the metaID and time.

An examples of some of the above is:

```
“/psialliance.org/Network/nodeUnreachable/1/10.5.13.204”,
“/psialliance.org/Network/connectionLost//216.74.33.9”,
“/psialliance.org/Network/networkOverrun/1”
```