Physical Security Interoperability Alliance
Service Model
Version 3.0
Revision 0
17 February 2015

Revision History	Description	Date	By
Version 1.0 Rev 0.1	Initial Draft	January 15, 2009	Frank Yeh
Version 1.0 Rev 0.9	Incorporated all Changes from Core Group Review	January 23, 2009	Frank Yeh
Version 1.0 Revision 1.0	Incorporated final changes from public comment period	February 13, 2009	Frank Yeh
Version 1.0 Revision 1.1	Incorporated negotiated changes from final review and ratification sessions	February 17, 2009	Frank Yeh
Version 1.0 Revision 1.2	Final Change for Versioning, Added in .xsd document	February 19, 2009	Frank Yeh
Version 1.1 Revision 1.0	Added mandatory PSIA as root of URL namespace; Corrected www.psia.org references to www.psialliance.org	April 15, 2009	Frank Yeh
Version 1.1, Revision 1.1	Updates for Service Model v1.1 in the following areas: Includes… • Adding implied :/PSIA" prefix to all URIs • Updated Service Discovery section to enable DNS-SD registration for routed networks. • Added Frank's Section for PSIA XML namespace definitions on the PSIA web site. • Added new required resource called '/PSIA/profile' to be used for identity management and node type and spec type support.; included XSD and example information. • Added the PSIA Common types XSD to the appendix • Updated/replaced service diagram with current resource tree definition..	January 27, 2011	Roger Richter
Version 1.2 Revision 1.0	• Added Chapter/Section 10 on Managed Data Transfer to standardize the way applications can rate/volume control transfer of large data objects. • Updated Section 13.1.6 "psiaCommonTypes.xsd" with new schema that adds support for new Area Control related common data types. Updates done by James Wang.	August 22, 2011	Roger Richter
Version 1.2 Revision 1.0b	• Clarification to Section 4.4 regarding 'Administrator' privilege management. • Clarification/update to Section 4.5 regarding use of PUT/POST to set configurations (GET is no longer an allowable method for 'sets' of config info). • Removed restriction on QSPs in	August 27, 2011	Roger Richter

	Section 7.5. We now allow QSPs and XML but discourage use of both on same resource. • In Section 7.6 removed incorrect statement about MIME XML payload type. • Section 13.1.6 'psiaCommonTypes.xsd' updated with new version containing the 'UID' definition.		
Version 1.2 Revision 1.0c	• Updated and simplified Section 4.5 to clarify the PSIA versus governmental requirements regarding the management of 'Administrator' accounts.	August 30, 2011	Roger Richter
Version 2.0 Revision 0.1	Moved the "/PSIA/System/"" Service and Resource hierarchy from the IPMD v.1.1 specification into this one. This is to make the 'System' hierarchy to all PSIA devices and systems. Please note that the "/PSIA/System/Video" and "/PSIA/System/Audio" resources were left in the IPMD spec due to their nature. Section/Chapter 9 has been completely updated.	Februar y 12, 2012	Roger Richter
Version 2.0 Revision 0.2	• Added Security explanation and qualifications to Section 9.2 regarding the offering of critical and/or destructive /System resources. They are only offered with 'Admin' permission levels per CSEC. • Made '/PSIA/System/firmwareUpdate' optional for non-embedded devices • Added Section 9.3.3 and 9.3.3.1 to enable a more flexible, intelligent way for updating executable images. • Updated 'requirements' tables in Section 9.1.2 to reflect new requirements that encompass all PSIA nodes..	March 26, 2012	
Version 2.0, Revision 0.3	• Deprecated and removed all requirements and references to HTTP 1.0. HTTP 1.1 is now the base. • Added new requirement in 'updateFirmware' and 'configurationData' to close all HTTP/TCP connections before rebooting to prevent 'dangling' connections. • Added Section 13 'Version Management of Functional APIs' to set the rules and guidelines for version management with respect to interoperability and migration.	April 10, 2012	R. Richter
Version 2.0 Revision 0.4	• Made minor edits based on review comments. • Major changes to Section 9.3.3 to create a new 'Update' service that is much more manageable than the prior	May 4, 2012	R. Richter

	'updateExecutables' resource. • Overhauled Section 9.3.3 to include 'state' and 'log' capabilities in the Update Service. Also, moved schedules reboot to the '/PSIA/System/reboot/time' resource.		
Version 2.0, Revision 0.4a	• In Section 10, removed the original definition of 'AFC.' Replaced with standard HTTP 'chunking' rules.	June 9, 2012	R.Richter
Version 2.0, Revision 0.5	• Added new PSIA Classification record to the SRC TXT record in Section 4.1. This now allows a client to read a DNS SRV record and determine what 'type' of device or system a PSIA node is. The spec codes from "profile.xsd" are used . • Edited "updateState.xsd" to get around unwarranted VS errors in Section 9.3.3.1. • Section 9.3.3.4 add new info to 'updateLog.xsd' to also record update status, and differentiated between exec and system files updated. • Clarified and normalized reboot options for '../configurationData', Section 9.3.4 and (now) for '../factoryReset', Section 9.3.5. • Fixed unresolved Section references. • Made several clarification edits.	July 2, 2012	R. Richter
Version 2.0 Revision 0.6	• Updates to /PSIA/Profile and supporting profile.xsd to support Operational Profiles • Changes to zerconf to support Operational Profiles with version numbers	July 20, 2012	J. Longo
Version 2.0 Revision 0.6a	• Made minor format changes to simplify and shorten (somewhat) the version tags for services/specs and profiles in the ZeroConf TXT record definitions. • Added version notation for services/specs in the ZeroConf TXT records similar to the profiles notation. • Provided new definition for 'Mandatory', and non-mandatory resource requirements in Section 9.2. • Updated and changed the mandatory '/PSIA/System' resources in Section/Chapter 9.2.2.	July 26, 2012	R.Richter
Version 2.0 Revision 0.6b	• Changed CSEC reference for Service Model v2.0 to require CSEC v1.1, not v1.0 as previously stated. • Updated Section 4.3 to deprecate and remove all references to Basic Authentication. Digest based authentication is now the requirement.	August 13, 2012	R.Richter
Version 2.0	• Changed Section 4.1 'Discovery' to	October	R.Richter

Revision 0.7	reflect implementation requirements by profile level. Discovery requirements are now broken into 2 profile levels: Basic and Full. • Added notation in the SRV text record for CSEC implementers to indicate the minimum transport security protocol implemented. • Section 9.2.2 Changed to reflect '/System/ requirements via profile levels. There are now Basic and Full profile requirements. All of the above were the result of the Systems and Compliance WG summit meeting.	31, 2012	
Version 2.0 Revision 0.7b/c	• Fixed minor typos in several sections. • Moved Optional/Dependent Resource requirements table for Storage after the requires resource tables in Section 9.2. • Added profiles columns to the /PSIA/System/Network resource requirements table in Section 9.2.2.	November 15, 2012	R,Richter
Version 2.0 Revision 0.7d	• Updated Section 4.3, "Authentication" to mirror the HTTP authentication requirements set by CSEC, • Also change the 'admin' login account requirement to be either a default or a configurable setting for device setup. The NULL password requirement was replaced with "password1234".	December 12, 2012	R.Richter
Version 2.0 Revision 0.8	• Fixed minor typos throughout • Section 4.1: Updated SRV TXT record example to meet the NVP format rules. Fixed SRV reference that should have been 'TXT'. • Updated XSD files for '/PSIA/System' resources to match the 'compiled versions.	Dec. 21, 2012	R.Richter
Version 2.0 Revision 0.8a	• Added a new profile for the Compliance WG as an extension to the Basic Profile in Section 9.2.2.	Jan. 17, 2013	R,Richter
Version 2.0. Revision 0.8b, Revision 0.9	• Minor typo fix to Section 5.2. • Added qualification to the /PSIA/Profile example, in Section 8.5, indicating that, in some cases, the newer PSIA schemas can create namespace conflicts with the original profile.xsd. Note for using an explicit namespace in the root element "PsiaProfile" now listed.	Aug.27. 2013; October 9, 2013	R.Richter
Version 2.1 Revision 0.1	• Added Section 9.10.1 /PSIA/System/Battery for battery mgmt..	Jan. 17, 2014	R.Richter
Version 2.1, Revision 0.2	• Added the new Section 9.10.1.1 Battery/unit resource to /PSIA/Battery.	Feb. 14, 2014	R.Richter
Version 2.1, Revision 0.3	• Changed the metric for battery 'charge' and threshold levels to be	Mar. 20, 2014	R.Richter

	either ‘percentage’ or ‘voltage’ based. • Section 9.10.1, ‘ChargeMetric’ is introduced to indicate of charge levels and thresholds are on percentage or voltage level format. • Section 9.10.1.1, Changed the threshold metrics to ‘floats’ to mirror the above metric types.		
Version 2.1, Revision 0.4	Added the Battery Event schema definition (XSD) to Section 14.1.8. This XSD file was moved here from the Common Metadata & Event Model spec.	May 23, 2014	R.Richter
Version 2.1, Revision 0.5	In Section 3 update the index as Optional.	Feb 11, 2015	Praveen Jha
Version 3.0 Revision 0.0	Topologies update	Feb 15, 2015	Jeffrey Longo

DISCLAIMER

Table of Contents

1. Introduction

The Service Model is intended to assist the PSIA working groups in creating new protocols or converting contributed protocols to a standard service model that will be common to all endorsed specifications. Adherence to this service model will ensure interoperability between compliant protocols.

This model is similar in nature to Web services but is geared towards lightweight computing requirements on devices. As such, these protocols will not use Simple Object Access Protocol (SOAP) as defined by the W3C-defined Web services but instead will use a simplified XML schema and/or xml schema documents (.xsd's).

Unless otherwise noted, all PSIA specifications should treat all configuration and management aspects as resources utilizing the REpresentational State Transfer (REST) architecture.

The PSIA Service Model is based on a REST architecture. While REST specifies that all interfaces are defined as resources, in the PSIA Model these resources are grouped by service. This architecture provides a convenient way to group related resources within a hierarchical namespace and lends itself to service discovery and future expansion.

The PSIA reserves the right to add services at any time provided said services adhere to the PSIA model as defined herein. Every effort should be taken to maintain full backward compatibility when adding new services. The PSIA Service Model is designed to support expansion with backwards compatibility.

2. PSIA Profiles and Topologies

PSIA Profiles define a set or subset of PSIA service requirements so that PSIA service providers and hosts can easily establish interoperability. A Profile is based off of one and only one PSIA functional specification, however they will identify what PSIA topology they adhere to as well as include an event streaming profile which by nature will identify further requirements from the common specifications. The common PSIA specifications are defined by the Systems Working Group and include of the Service Model, CSEC, and Common Metadata and Event Model (CMEM).

PSIA identifies the following topologies:

- Master-Slave over the Internet – Configuration, control, and data exchange between a higher-level management system ("master"), and a lower-level device or system ("slave"). The master system is responsible for configuring and controlling the slave system, including sending commands. The slave system generally reports its events and status to the master, which may be managing multiple slaves and aggregating this data. The master and slave are separated by multiple router and/or firewall boundaries and are generally considered to talk over the Internet.
- Master-Slave over a LAN – The same as Master-Slave over the Internet, however the master and slave are located behind a common firewall and are otherwise considered to be on the same Local Area Network (LAN).

- Peer to Peer[1] over the Internet: Data exchange between 2 devices/servers at the same level of the system architecture, for example, between 2 access control panels, between an access control and an intrusion panel, between one of these panels and a video device, etc. The data shared is often events, point status, and the invocation of commands. The master and slave are separated by multiple router and/or firewall boundaries and are generally considered to talk over the Internet.
- Peer to Peer over a LAN – The same as Peer to Peer over the Internet, however the peers are located behind a common firewall and are otherwise considered to be on the same Local Area Network (LAN).

3. Design Considerations

3.1. 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.

1 Since PSIA utilizes a client-server design, this really refers to the idea that both sides are peers in the hierarchy of devices and servers within a system, rather than being peers at the protocol level. Note that it is also possible for both devices to be clients and servers at the same time, which is like a "double" peer-to-peer. The term peer-to-peer in this document generally refers to the simpler configuration, with one peer as a client and the other as a server.

3.2. Conformance

Every PSIA protocol specification will define one or more PSIA compliant services. To ensure interoperability, the following conformance requirements are also implied in each PSIA specification.

Minimum API Set

The “Minimum API Set” is determined by the requirements of the PSIA Profile being implemented.

XML Requirements

A system/device must support the syntax as defined by the W3C XML 1.0 specification.

A system/device must support the UTF-8 character set as described by

http://www.w3.org/International/O-charset

Additionally, XML content must correspond to the following Schemas as defined in Appendix 10:

- “ResourceList XML Schema”
- “ResourceDescription XML Schema”
- “QueryStringParameterList XML Schema”
- “ResponseStatus XML Schema”

Vendors may optionally extend this standard to include proprietary XML content as long as it does not conflict with the minimum set of APIs. If the PSIA specification does not provide explicit extensibility tags, it is recommended to use a vendor-specific XML namespace to avoid conflicting names that may arise with future revisions.

For example, if vendor XYZ123 Inc intends to extend the XML standard to include a <configOption> parameter, it is recommended to use <configOption xmlns=”urn:XYZ123-com:configuration:options”> to avoid future namespace conflicts.

Protocol Requirements

A system/device must support transport of XML via HTTP/1.1 protocol as specified in RFC2616. HTTP/1.1 is used in order to support key features (persistent connections, HTTPS, etc.).

3.3. HTTP Methods and REST

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.

3.4. HTTP Status Codes and REST

The following table shows how the HTTP status codes map to REST operations along with the general use case for response headers and bodies. For more information, please see the table under each REST API.

HTTP Status Codes	REST Meaning	POST	GET	PUT	DEL
200	"OK" - The request has succeeded. Header Notes: None Body Notes: The requested resource will be returned in the body.		X	X	

HTTP Status Codes	REST Meaning	POST	GET	PUT	DEL
201	"Created" - The request has created a new resource. Header Notes: The *Location* header contains the URI of the newly created resource. Body Notes: The response returns an entity describing the newly created resource.	X			
204	"No Content" - The request succeeded, but there is no data to return. Header Notes: None Body Notes: No body is allowed.			X	X
301	"Moved Permanently" - The requested resource has moved permanently. Header Notes: The *Location* header contains the URI of the new location. Body Notes: The body may contain the new resource location.		X		
302	"Found" - The requested resource should be accessed through this location, but the resource actually lives at another location. This is typically used to set up an alias. Header Notes: The *Location* header contains the URI of the resource. Body Notes: The body may contain the new resource location.		X		
400	"Bad Request" - The request was badly formed. This is commonly used for creating or updating a resource, but the data was incomplete or incorrect. Header Notes: The Reason-Phrase sent with the HTTP status header may contain information on the error. Body Notes: The response may contain more information of the underlying error that occurred in addition to the Reason-Phrase.	X	X	X	

HTTP Status Codes	REST Meaning	POST	GET	PUT	DEL
401	"Unauthorized" - The request requires user authentication to access this resource. If the request contains invalid authentication data, this code is sent. Header Notes: At least one authentication mechanism must be specified in the *WWW-Authenticate* header. The Reason-Phrase sent with the HTTP status header may contain information on the error. Body Notes: The response may contain more information of the underlying error that occurred in addition to the Reason-Phrase.	X	X	X	X
403	"Forbidden" - The request is not allowed because the server is refusing to fill the request. A common reason for this is that the device does not support the requested functionality. Header Notes: The Reason-Phrase sent with the HTTP status header may contain information on the error. Body Notes: The response may contain more information of the underlying error that occurred in addition to the Reason-Phrase.	X	X	X	X
404	"Not Found" - The requested resource does not exist. Header Notes: None Body Notes: None	X	X	X	X
405	"Method Not Allowed" – The request used an HTTP method that is not supported for the resource because the {API Protocol} specification does not allow this method. If the device does not support the functionality but it is a valid {API Protocol} operation, then a 403 is returned. Header Notes: The *Allow* header lists the supported HTTP methods for this resource. Body Notes: None	X	X	X	X
409	"Conflict" – The operation performed conflicted with an internal state or a process being performed. This is a transient condition and the operation can be retried at a later time.		X	X	X

500	"Internal Server Error" - An internal server error has occurred. Header Notes: None Body Notes: None	X	X	X	X

HTTP Status Codes	REST Meaning	POST	GET	PUT	DEL
503	"Service Unavailable" – The HTTP server is up, but the REST service is not available. Typically this is caused by too many client requests. Header Notes: The *Retry-After* header suggests to the client when to try resubmitting the request. Body Notes: None	X	X	X	X

3.5. Unique Identifiers

IDs are defined as URL-Valid Strings, as required by REST. The device will create an ID for all resources that add a resource via a POST request. The host can specify ID numbering by using PUT exclusively for creation of resources.

In some topologies a globally unique IDs are required. Globally unique IDs should be derived using the method described in RFC4122.

3.6. ID Encoding

Because IDs will occur as part of a URI, there are two ways to encode an ID: either following RFC 3986 or, for pure binary IDs, as a hex string

RFC 3986 first converts the URI to UTF and then prints the following unreserved characters in the URI without any encoding:

- A-Z
- a-z
- 0-9
- -
- .
- _
- ~

All non-printable or reserved characters will be encoded as a two digit hex value prefixed by a %. For example, a space (ASCII value of 32) will be encoded as %20.

Because a pure binary ID can contain values that might interfere with the operation of browsers and web servers, PSIA protocols support hex encoding of the ID. The ID must begin with 0x (0X is also acceptable) followed by pairs of hex values. Each hex pair represents a single byte in the ID. For

example: 0x3F431245DE67FAC46F9D034CA23AEFD4. The hexadecimal characters A-F can also be represented by a-f. So 0x3f431245de67fac46f9d034ca23aefd4 is equivalent to the previous ID.
If readable IDs are desired, it is recommended that IDs are created with unreserved, printable ASCII characters.

4. Architecture and Namespace

In a typical REST-based namespace, every node or object in the tree-structured hierarchy is considered a resource.

The PSIA model adds a resource sub-class called “Service”. Services are simply nodes which can contain other nodes. Nodes that do not contain other nodes (other than the standard node resources of the PSIA model) will continue to be called resources, while the term node will be used to refer to both services and resources.

Viewed as a tree, services are analogous to branches and resources are analogous to leaves.

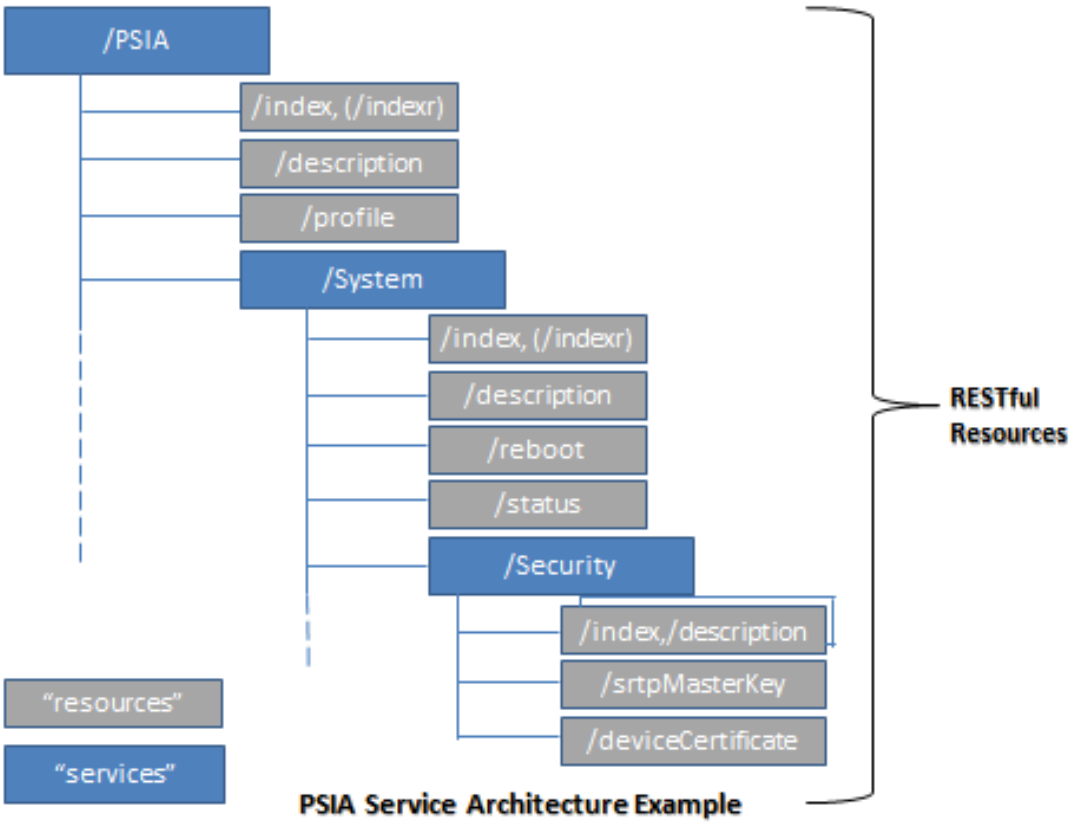
Each Service and branch node must contain the following standard PSIA resources:

description	which will respond to an HTTP GET with a ResourceDescription datablock
index	which will respond to an HTP GET with a ResourceList datablock

Each node may contain the following standard PSIA resources:

indexr	which will respond to an HTTP GET with a ResourceList datablock
capabilities	which will respond to an HTTP GET with a resource-specific XML Document

The index resource will return a list of all the immediate “children” of a node. For services, this list could contain other services as well as resources. For resources, this list should only indicate which standard PSIA resources (IE description, index, and optionally indexr and capabilities) are contained. The optional indexr resource will return a recursive listing that descends through the namespace hierarchy.



PSIA Service Architecture Example

Resource Name	Description
Description	will respond to an HTTP GET with a <ResourceDescription> datablock
Capabilities	will respond to an HTTP GET with a resource-specific XML Document
Index	will respond to an HTTP GET with a <ResourceList> datablock
Indexr	will respond to an HTTP GET with <ResourceList> datablock

For all PSIA protocols, the root namespace of "PSIA" is mandated, meaning it has to be included in the URL. Therefore, the root of any PSIA service's namespace will be "PSIA".

Each service and resource will be mandatory or optional as specified by the topology, indicating to implementers which services they must implement at a minimum.

Service or Resource URL	Description
/PSIA/System	Resources related to general system configuration and operation
/PSIA/profile	V1.1 Resource that describes the functional identity and manageable attributes of a node.

/PSIA//System/Storage	Resources related to local storage
/PSIA/System/Network	Resource related to Network settings
/PSIA/Security	Resources related to security of the device (deprecated)
/PSIA/Security/AAA	Resources related to AAA functions (deprecated)
/PSIA/CSEC	V1.1 PSIA Common Security Model (CSEC) resources used for systemic security management. This Service, and its resources, deprecate the older 'PSIA/Security' resources.
/PSIA/Metadata	V1.2 PSIA Common Metadata/Event Model resources. This Service, and its associated resources specify the generation, transport, consumption, and processing of all non-audio/video information. This includes all event, statistical, reporting, alarm, and logging information.
/PSIA/Streaming	Resources related to streaming media content
/PSIA/PTZ	Resources related to Pan/Tilt/Zoom
/PSIA/Archive	Resources related to storage of content
/PSIA/Diagnostics	Resources related to Diagnostics
/PSIA/Custom	Resources that are specific to a protocol or vendor specific

3.1 Multiple Channels and Versions

To provide for multi-channel support, a service must insert the implied "Channels" service as a child-node which should then contain an ID resource for each channel. Each ID resource will then respond to each of the resources applicable to the service.

For Single-Channel Devices, the Channels service must still be included to maintain consistency between single and multi-channel devices and to provide for the case where a multichannel device has only created a single channel.

Note that Channel IDs are arbitrarily assigned by the device.

(EG. For a single channel device:
/Streaming/Channels/0/keyFrame
For a multi-channel device
/Streaming/Channels/0/keyFrame
/Streaming/Channels/1/keyFrame)

Devices may either pre-define this multichannel structure or support dynamic additions and deletions of channels (using HTTP POST and DELETE) as applicable.

In order to differentiate services that essentially provide for multiple instances of something within the hierarchy, it is recommended that services at the root level be referred to as "Root Services" while the term service continue to be used to describe any node that contains other nodes (EG Streaming is a Root Service, Channels is not).

Each node, be it a resource or service, will be able to return a description of itself within the service model. This description will include a version attribute to support versioning within the PSIA Service

model. While this practice will allow resources with different versions to exist within the same services, it is mandatory that all resources within a service container are fully backward compatible.

If a new service version is introduced that does not maintain backwards compatibility with previous versions, then a new service must be created for the new, incompatible version (EG /Streaming and /StreamingV2). IE it is acceptable to add resources to a Service but not to replace them with new versions that are not backward compatible. If new resource versions must be added, the Root Service name should be changed to indicate a new Service version.

5. System Flow

Before any protocol can be used to work with a device, it may need to be discovered. It is required that the ZeroConf (Zero Configuration Networking) technology be supported to discover/locate devices for LAN-based topologies. Once this step is accomplished, transactions can commence. For Internet-based topologies, manual addressing should be used, or the implementers can optionally implement DNS-SD. ZeroConf is normally expected to operate in a local area network.

HTTP requests are made through the device's web server. The HTTP response may contain XML content (for GET actions), XML response information (for PUT or POST actions), or various text/binary content (for retrieval of configuration data, etc.). Edge devices should be able to handle overlapping/simultaneous HTTP requests, as well as persistent connections to handle multiple HTTP transactions.

The XML content should be described by .xsd documents. Relevant XML data structures must be documented in an Appendix section of each PSIA Specification.

5.1. Service Discovery

ZeroConf (Zero Configuration Networking) technology specifies the mDNS (Multicast DNS) and DNS-SD (DNS Service Discovery) protocols, as described in "http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt" as the mechanisms to discover/locate services and devices on an IP network. All LAN-based PSIA topologies require mDNS for node discovery. DNS-SD can optionally be used for service discovery however it is not presently required in any PSIA topology. To support this discovery model, the PSIA has registered a DNS SRV (RFC 2782) service type to be used to discover all PSIA nodes via mDNS and/or DNS –SD. Note that when mDNS discovery is required via a topology, only discovery is required - ZeroConf's use of Local IP address allocation is optional.

Please note that both mDNS and DNS-SD discovery protocols may be enabled/disabled by nodes that implement the "/PSIA/System/Network/interfaces/<id>/discovery" resource interface. If no configuration interface is implemented, then the profile level determines the protocols that are to be active by 'default.'

DNS-SD Registration

PSIA nodes implementing DNS-SD do the following:

- Attempt to register its PSIA-defined service record (i.e. SRV) using the "psialliance.org" domain with each DNS Server listed in its IP configuration using DNS-SD. This enables discovery across subnet boundaries.

- Irrespective of whether the prior DNS-SD SRV registrations pass, or fail, each device is required to also honor mDNS requests for its service type definition in the 'local' domain.
- Please note that different tools and APIs affect how the above operations are accomplished programmatically. The goal is to have ALL PSIA service providers register their devices in the "psialliance.org" domain for routed network support, and also, concurrently, support local mDNS discovery.

The format of a PSIA compliant Content Mgmt device's SRV must always start with the DNS SRV 'services' and 'protocol' prefixes of : "***_psia._tcp."***, followed by the other pertinent information as outlined in RFC 2782, and in the following paragraph.

DNS-SD discoveries, initiated by entities seeking PSIA devices and services (i.e. clients, management servers, etc.), should use the PSIA's public DNS service type to discover the device according to DNS Service Discovery (http://www.dns-sd.org/ServiceTypes.html). Once a device is established as a PSIA-compliant device, its services and resources can be discovered using standard HTTP GETs using the standard, mandatory REST resources.

The following information should be advertised in the SRV record:

- A (REST) path of "/PSIA/index" – can be obtained from the "path" key in the TXT record
- The {host} – via an IP address or domain name, can be obtained from the service's SRV record
- The {port} – can be obtained from the service's SRV record
- The version of the DNS SRV record in "txtvers"
- The PSIA Service protocol version in "protovers"
- The PSIA classification of a node, via the tag definition of the specifications implemented, MUST be listed in the TXT record. Additionally, if a PSIA node supports a PSIA profile, this MUST be listed also.The notation is:
    - Each list starts with the 'list specifier tag'. The tags are:
        - "psia.svcs=" for the PSIA specifications/services list; and
        - "psia.profiles=" for the PSIA profiles list (where implemented).
    - The list(s) MUST begin with an opening square bracket ("[") and end with a closing square bracket ("]").
    - The PSIA specification/services list elements are each comprised of a specification tag from the "Profile.XSD" schema "PsiaSpecType", defined in Section 14.5.12.
    - Each PSIA Profile element tag is published by the PSIA Profiles specification.
    - All Services and Profile tag element MUST also provide the accompanying version number by appending a 'slash' ("/") character to the associated element tag, followed by the version number (e.g. "csec/1.0", "enterpriseAccessControl/1.0"). SPECIAL NOTE: CSEC implementers MUST also indicate the minimum transport security protocol level they are compliant with; i.e. if a node implemented SSLv3 and TLSv1.0 it would indicate "(sslv3)" as is its minimum level). This session security level tag follows the CSEC/Version tag surrounded by parentheses. Examples follow.
    - An example of a PSIA IP Media Device that supports PSIA Video Analytics, PSIA CMEM-based event generation, and CSEC v1.1 follows:
        - "psia.svcs[ipmd/1.1,videoAnalytics/1.0,cmem/1.2,csec/1.1(sslv3)]"
    - An Example of a Control Panel that implements the PSIA Area Control spec and the Enterprise Profile for Area Control is:
        - "psia.svcs=[actl/1.0,cmem/1.2,csec/1.1(tlsv1.0)]"
        - "psia.profiles=[enterpriseAccessControl/1.0]"

Once a PSIA-compliant device has been discovered, an HTTP GET of its mandatory '/index' resource returns a list of the services that the device supports. At this point, the standard methods are used to "walk" the namespace tree and discover the supported services and resources.

It should be noted that the "index" resource returns only the first level resources of a node, but the "/indexr" resource will return a recursive tree structured list with the current resource as root.

It should be noted that the index resource returns only the first level resources of a node, but the indexr resource will return a recursive tree structured list with the current resource as root. Additionally, the "/PSIA/profile" resource, described in Section 8,5, indicates the type of PSIA node detected, its identity, and the number, type and revision of the PSIA specifications supported by that node.

5.2. Persistent Connections

All PSIA systems and devices should support persistent connections in order to support video management systems or client applications that issue multiple HTTP(S) transactions. The PSIA assumes that HTTP/1.1 is implemented and utilized according to RFC2616.
A video management system or client application should, when using a persistent connection for multiple transactions, implement the "Connection: Keep-Alive" HTTP header. The management system should also use the "Connection: close" HTTP header field for the last transaction made within this persistent connection. This process assumes that the application is aware of the last request in a sequence of multiple requests.

5.3. Authentication

When an application sends any request to the device, HTTP sessions must be authenticated by means of Digest authentication according to RFC 2617. HTTPS session may use either Basic or Digest based authentication. This means the user access credentials are sent along with each request. If a user is authenticated, the request will follow the normal execution flow.

Example client HTTP request header and body with no authentication credentials:

```
GET /PSIA/index
…
```

Example unauthorized HTTP response header and body:

```
HTTP/1.1 401 Unauthorized
…
WWW-Authenticate: Digest realm="testrealm@host.com",
                        qop="auth,auth-int",
                        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                        opaque="5ccc069c403ebaf9f0171e9517f40e41"
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx    (note: xxx = size of XML block)

<?xml version="1.0" encoding="UTF-8" ?>
<ResourceList version="1.0" xmlns="urn:psialliance-org:resourcelist">
     …
</ResourceList>
```

Example client HTTP request header and body with authentication credentials (username "Mufasa" and password "Circle of Life"):

```
GET /PSIA/index
…
Authorization: Digest username="Mufasa",
                      realm="testrealm@host.com",
                      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                      uri="/dir/index.html",
                      qop=auth,
                      nc=00000001,
                      cnonce="0a4f113b",
                      response="6629fae49393a05397450978507c4ef1",
                      opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Example authorized HTTP response header and body:

```
HTTP/1.1 200 OK
…
-Type: application/xml; charset="UTF-8"
Content-Length: xxx    (note: xxx = size of XML block)

<?xml version="1.0" encoding="UTF-8" ?>
<ResourceList version="1.0" xmlns="urn:psialliance-org:resourcelist">
     <Resource>
          …
     </Resource>
</ResourceList>
```

5.4. Access Restrictions

All supported resources on a device must be fully accessible to users with the "Administrator" privilege level. This means that in order to use the full suite of resources a device offers, authentication must be granted with a user account having a privilege level corresponding to "Administrator".

There are no restrictions as to which resources are accessible to users with other privilege levels. A vendor may choose to limit, for example, the allowable resources for user accounts with lower privileges. However, since user-specific authorization is not a function of the protocol, it is often assumed that full administrative rights will be available via the protocol. User-specific authorization functions are expected to be handled by the calling application.

5.5. Setting Configurations

Resources to *set* device configurations will use the HTTP PUT/POST methods.. If device configuration parameters are conveyed via an XML payload, the inbound XML format is defined according to a resource-specific XML schema For PUT/POST operations, the request status will be indicated by the XML response information returned from the device, and can be used to indicate the status of the set operation. This XML format is defined according to "XML Response Schema" (see section 4.12 for details). After successfully updating the repository, the device returns an XML response with status code "OK". A separate status code is used for unsuccessful operations. In either case, the device will not return a response until it is ready to continue normal operation – this includes accepting streaming requests, receiving behavioral control commands, etc.

Example HTTP request header and body:

```
POST /PSIA/System/deviceInfo HTTP/1.1
…
Content-Type: application/xml; charset=”UTF-8”
Content-Length:xxx    (note: xxx = size of XML block)

<?xml version="1.0" encoding="UTF-8" ?>
<DeviceInfo version="1.0" xmlns="urn:psialliance-org:system:deviceinfo">
…
</DeviceInfo>
```

Example HTTP response header and body:

```
HTTP/1.1 200 OK
…
Content-Type: application/xml; charset=”UTF-8”
Content-Length: xxx    (note: xxx = size of XML block)

<?xml version="1.0" encoding="UTF-8" ?>
<ResponseStatus version=“1.0” xmlns=“urn:psialliance-org:response”>
…
</ResponseStatus>
```

5.6. Getting Configurations

Resources to get device configurations or status information will use the HTTP GET method. If successful, the result will be returned in XML format according to the resource description. If the request is unsuccessful for any reason (i.e. not authenticated), the result will be returned in XML format according to “ResponseStatus XML Schema”. The Content-Type and Content-Length will be set in the headers of the HTTP response containing the XML data. The Content-Type is: application/xml; charset=“UTF-8”.

Example HTTP request header and body:

```
GET /PSIA/System/deviceInfo HTTP/1.1
…
```

Example HTTP response header and body:

```
HTTP/1.1 200 OK
…
Content-Type: application/xml; charset=”UTF-8”
Content-Length:  xxx    (note: xxx = size of XML block)

<?xml version="1.0" encoding="UTF-8" ?>
<DeviceInfo version="1.0" xmlns="urn:psialliance-org:system:deviceinfo">
    …
</DeviceInfo>
```

5.7. Getting Capabilities

Capabilities can also be retrieved by any resources node that specifies an XML payload for inbound data with an HTTP GET of its “capabilities” resource. In other words, a client application can query a device for its capabilities in order to understand what XML tags are supported, the acceptable data ranges, etc. See Section 5.4 for more detail on the returned capabilities.

Example HTTP request header and body:

```
GET /PSIA/PTZ/channels/ID/0/absolute/capabilities HTTP/1.1
…
```

Example HTTP response header and body:

```
HTTP/1.1 200 OK
…
Content-Type: application/xml; charset=”UTF-8”Content-Length: xxx (note: xxx = size of XML
block)
<?xml version="1.0" encoding="UTF-8" ?>
<PTZData version=“1.0” xmlns=“urn:psialliance-org”>
    <pan min=”-100” max=”100”/>
    <tilt min=”-100” max=”100”/>
    <zoom min=”-100” max=”100”/>
    <Momentary>
        <duration min=”0”/>
    </Momentary>
    <Relative>
        <positionX min=”0” max=”1024”/>
        <positionY min=”0” max=”1024”/>
        <relativeZoom min=”-100” max=”100”/>
    </Relative>
    <Absolute>
        <elevation min=”-90” max=”90”/>
        <azimuth min=”0” max=”360”/>
        <absoluteZoom min=”0” max=”100”/>
    </Absolute>
    <Digital>
        <positionX min=”0” max=”1024”/>
        <positionY min=”0” max=”1024”/>
        <digitalZoomLevel min=”0” max=”100”/>
    </Digital>
</PTZData>
```

5.8. Uploading Data

Resources to upload data (i.e. firmware, executables, system files, configuration data/files, etc.) to the device will use the HTTP PUT method for existing file, and the POST method for new files. The content of the data will be stored in the body of the HTTP request. The Content-Type and Content-Length will be set in the headers of the HTTP request. The Content-Type MUST be set to the applicable data transfer type. Additionally, the use of HTTP ‘chunking’ is recommended for data transfers of 8 Kilobytes, or more.
Example HTTP upload request header and body:

```
POST /PSIA/System/configurationData HTTP/1.1
…
Content-Type: application/ xml; charset=”UTF-8”

[proprietary configuration data content]
```

Example HTTP upload response header and body:

```
HTTP/1.1 200 OK
…
Content-Type: application/xml; charset=”UTF-8”
Content-Length: xxx    (note: xxx = size of XML block)

<?xml version="1.0" encoding="UTF-8" ?>
<ResponseStatus version=“1.0” xmlns=“urn:psialliance-org:response”>
     …
</ResponseStatus>
```

5.9. Receiving Data

Resources to receive data (i.e. configuration file, etc.) from the device will use the HTTP GET method. The content of the data will be stored in the body of the HTTP response. The Content-Type and Content-Length will be set in the headers of the HTTP response, according to the type of data being returned.
The client may use the Accept: header string to tell the server what formats it accepts. Depending on what the client accepts, the server may transcode, transform or even compress the data to match the client’s expectations.

Example HTTP download request header and body:

```
GET /PSIA/System/configurationData HTTP/1.1
…
```

Example HTTP download response header and body:

```
HTTP/1.1 200 OK
…
Content-Type: application/octet-stream
Content-Length: xxx    (note: xxx = size of XML block)

[proprietary configuration data content]
```

5.10. Operations

For stateless operations (i.e. function calls) the formula is:

PUT /Service/<Operation>

Resources must indicate in their descriptions which XML payload is required or the query string parameters to be used in the operation.

5.11. Diagnostics

Diagnostics (and other stateful operations) run in the background on the device, so it must be possible to create them asynchronously and be able to query their status.
The REST model works well here:

Request	Result
POST /PSIA/Diagnostics/<command>	Returns diagnostic ID
GET /PSIA/Diagnostics/<command>/<ID>	Get information on this ID

DELETE /PSIA/Diagnostics/<command>/<ID>	Delete command in progress
GET /PSIA/Diagnostics/commands	Get information on all commands running

5.12. Response Status

Responses to many resource calls contain data in the form of the ResponseStatus XML document. Within each specification, separate services and resources may each have their own data structures. The only provision of the model is that each ResourceDescription must indicate which structures are used and each structure must be defined in an XML schema document within the specification document. If resources do not define their own response structures, they may use the PSIA standard ResponseStatus structure as defined in Appendix 10.

Status Code

A ResponseStatus with statusCode=OK will be sent after the command has been completely processed on the device. Even if the request contains some parameters that are not supported, the device will ignore those parameters and return statusCode=OK.

A device will send a Device Busy response to a command which cannot be processed at that time (eg. receiving a reboot command while the flash is being updated)

If the device fails to perform the request - possibly due to a hardware error - it will return a Device Error statusCode and a fault message in the statusString.

An Invalid Operation statusCode is returned in response to a command that has not been implemented. Invalid Operation is also returned if an authentication attempt fails or the logged in user has insufficient privileges to execute the command.

An Invalid XML Format statusCode is returned if the XML is badly formed and causes the parser to fail. The statusString should indicate the fault.

An incomplete message or a message containing an out-of-range parameter will return an Invalid XML Content statusCode and associated statusString.

For settings that require a reboot to take effect, such as changing the network address or a firmware update, the Reboot Required statusCode is returned.

Status String

It is recommended that for all responses where the returned statusCode is not OK, a descriptive statusString be returned indicating the reason the command was not completed.

5.12.1. ID

In POST operations where the device will return an ID of the resource created, this attribute will be used to pass back the created ID. In Service Model v2.0, and later, this ID MUST be either in ‘LocalID’

format (UTF-8 decimal 'unsigned int' string), or in "GlobalID" format (IEC 9834-8/ITU X.667 UUID/GUID). ***Of special note is the fact that UUIDs/GUIDs are NEVER conveyed in URLs with the starting and ending 'curly braces' (i.e. "{" and "}", respectively)***. Please reference "psiaCommonTypes.xsd" for the format of these common PSIA values.

5.13. Processing Rules

Any field (particularly in the inbound XML parameters) that is not supported by the device should be ignored. For any given resource there may be some special processing rules. These rules are documented in the column associated with the heading "Implementation Note".

6. XML Modeling

6.1. File Format

All XML files must use UTF-8 (8-bit UCS/Unicode Transformation Format) encoding according to RFC3629. A BOM (byte-order mark) can optionally be used. Thus, a media device should support UTF-8 encoding with or without a BOM.

6.2. Data Structures

Any Resource can specify separate input and output XML Documents. If a specific data structure is defined, these must be specified as XML Schema Documents (xsd) within the specification. The xsd's created for PSIA specifications are to be included in the appendix section of the relevant specification. In addition, the PSIA will be posting xsd documents of relevant schemas at http://www.psialliance.org/schemas to support online reference of the schemas. However, there is no guarantee that the schemas will be posted at the same time the documents are published. For this reason, the schema definitions within the specification documents themselves are the minimal requirement.

6.3. Lists

Many of the XML blocks contain lists. The syntax of these lists is <XXXList>, where XXX is a name referring to the XML setting. Inside of the <XXXList> tag is one or more <XXX> nodes. As an example, the <ChannelList> block may contain content as such:

```
<ChannelList>
    <Channel>
            <id>1</id>
            …
    </Channel>
    <Channel>
            <id>2</id>
            …
    </Channel>
    …
</ChannelList>
```

6.4. Capabilities

Capabilities for any resource that defines an XML block for input will be returned as an XML document that is essentially an XML instance of the resource-specific input XML block. This XML document must contain the acceptable values for each attribute.

While XML Schema Documents are also required of any XML data defined by any PSIA specification and xsd documents are capable of defining the acceptable range of values for any attribute, using a global xsd to define capacities would imply that all devices support the same options for any parameter. By allowing devices to respond to the capabilities request, each device can support different values for any attribute, within the constraints of the schema.

Capability Attribute	Description	Syntax	Applicable XML Data Types
Min	The minimum character length for a string, or the minimum numerical value of a number	Examples: min="0" min="64" min="-100" (numerical only) min="1.2"	All except fixed data types [1]
max	The maximum character length for a string, or the maximum numerical value of a number	Examples: max="5" max="64" max="4096" max="10.50"	All except fixed data types [1]
range	Indicates the possible range of numerical values within the "min" and "max" attributes of an element. This attribute should only be used if the possible values for an XML element does not include the entire numerical range between "min" and "max" attributes	Ranges are listed in numerical order separated by a "," character. A range has the form "x~y" where x is the range floor and y is the range ceiling. Single numbers may also be used. *Example*: if an XML element supports values 0, 123, 1024 to 2000, and 2003, the syntax would be: range="0,123,1024~2000,2003"	All numerical data types
opt	Lists the supported options for a CodeID data type. Required for XML elements with a CodeID data type. This attribute should *not* be used for any other data type	If all options are supported, the syntax is "all". Otherwise, supported options are listed separated by a "," character. Examples: opt="all"	CodeID

		opt="1,2,3" opt="1,2,5,8,9,10,11"	
Def	Indicates the default value of the XML element. If the element has no default value, this attribute should *not* be used	Examples: def="1234" def="Device ABC" def="3"	All data types
reqReboot	Indicates if configuration of this XML element requires a device reboot before taking effect. If an element doesn't require a reboot, this attribute should *not* be used	reqReboot="true"	All data types
dynamic	Indicates if an XML element has dynamic capabilities dependent on other XML configurations. For example, if an element's data range changes based on another element's configured value, this attribute must be used. In this case, the element's capability attributes must always reflect the current device configuration	dynamic="true"	All data types
Size	Indicates the maximum number of entries in an XML list. This attribute is only applicable to XML list elements. This attribute should not be used for any other type of element (see section 6.3 for details)	*Example*: If a device supports 5 users the example would be <UserSetting> <UserList size="5"> …	Only supported for list elements (see section 6.3)

[1] Fixed, pre-defined data types do not need certain capability attributes because their formats/data ranges are already defined. Where pre-defined data types are used, each protocol document must include an enumeration of these formats in an Appendix section.

7. Custom Services & Resources

In order to support system/device specific resources that are not common to the public service definitions, the CUSTOM service type is provided. An HTTP GET of the index resource of the CUSTOM service returns a list of the custom services and resources supported by the system/device.

For each custom resource, an implicit mandatory resource named “Description” must be supported. An HTTP GET of any custom resource’s Description resource must return a ResourceDescriptionBlock similar to the Resource Description information described in section 7.6.

Custom services and resource can be used to support protocol-specific resources that are thought to be of an interim nature (IE a forthcoming protocol will most probably deprecate these resources) or vendor-specific proprietary resources. As long as all custom services and resources are implemented according to the PSIA Service Model, they can be discovered and called by PSIA-compliant clients and applications.

8. Interface Design

The HTTP URL format is of the general form

```
<protocol>://<hostname>:<port>/<URI>? P1=v1&P2=v2....&pn=vn
```

All requests will follow this format. A brief description of these components follows:

8.1. Protocol

The protocol field refers to the URL scheme that will be used for the particular request. Note that the current specification allows the following schemes:

- http
- https

8.2. Hostname

The hostname field refers to the hostname, IP address, or the FQDN (fully qualified domain name) of an IP device.

8.3. Port

The port field indicates the port number to be used for the HTTP request. The default port number for HTTP is 80. For HTTPS, the default port is 443. For RTSP, the default port is 554. If neglected in the URL, these default port numbers will be used for the request (as defined in RFC2616, RFC2818, and RFC2326 respectively).

The HTTP and HTTPS port number is configurable for IP devices. The standard HTTP and HTTPS ports (80 and 443) will be assumed unless otherwise specified.

8.4. URI

The URI absolute path is most often of the form "<SERVICE>/<resource>" where <resource> corresponds to one of the resources defined in the specification. For example, <SERVICE> could refer to "System" or "Security". This is true for resources that update or retrieve device configurations.

8.5. Query String

Resources specify required and optional query string parameters. In either case, these query string parameters must be listed in name-value pair syntax (p1=v1&p2=v2…&pn=vn) following the URI.

Example GET HTTP request with query string parameters:

```
GET /PSIA/Streaming/Channels/1/picture?snapShotImageType=jpeg
…
```

Example POST HTTP request with query string parameters:

```
PUT /PSIA/System/time?localTime=2009-02-16%2013:30:00
…
```

Each resource may define a set of parameters, in the form of name-value pairs, which exist in the query string. Resources may define the use of query string parameters and XML payloads as methods for conveying transaction parameters; however, the use of both for the same resource is discouraged as a practice.

8.6. Resource Description

For each resource in this document, the following components are defined:

Format – indicates the URL format of the HTTP request

Type – indicates whether this is a service or resource

Method specific (GET, PUT, POST, DELETE)

Query string parameters – indicates the name/value pairs (P1,P2,P3,…Pn) for the resource.

Inbound Data– indicates inbound data for the resource as follows:

- **NONE** – indicates no input data
- **DataBlock** - the name of a Data Block defined within the specification. Datablocks used here must be defined within the specification document. In addition, it is strongly recommended that .xml schema documents be created for each referenced datablock.
- **Mime type** – indicates that the input data is in the HTTP payload with the indicated mime type.

If a device doesn't support particular XML tags or blocks, they need not be used in the resource operations.

Generally, if fields are not provided in the inbound XML, the current values for these fields should remain unchanged in the device's repository.

If required fields do not already exist in the device's repository, they must be provided in applicable resource operations.

Function – describes the general function behavior

Return Result – describes the response from the HTTP request

Implementation Note – describes the implementation behavior and any special processing rules for the resource.

For example,

URI	Index	Version	1.0	Type	Resource
Function	Enumerate child nodes				
Methods	Query String(s)	Inbound Data		Return Result	
GET	None	None		<ResourceList>	
Notes	Returns a flat (non-recursive) listing of all child nodes				

In order to support discovery of CUSTOM service resources, this resource description data structure is also captured as a data block of type ResourceDescription. Whenever an HTTP GET of a device's CUSTOM/Index resource is executed, a list of the device's custom resources is returned. For each custom resource, an HTTP GET of the mandatory resource "Description" will return a ResourceDescription Block indicating what the resource does and how it should be used.

9. PSIA Standard Resource Descriptions

This section describes the standard, common PSIA resources

9.1. index

URI	index	Version	1.0	Type	Resource
Function	Enumerate child nodes				
Methods	Query String(s)	Inbound Data		Return Result	
GET	None	None		<ResourceList>	
Notes	Returns a flat (non-recursive) listing of all child nodes				

9.2. indexr

URI	indexr	Version	1.0	Type	Resource
Function	Enumerate child nodes				
Methods	Query String(s)	Inbound Data		Return Result	
GET	None	None		<ResourceList>	
Notes	Returns a recursive listing of all child nodes				

9.3. description

URI	Description	Version	1.0	Type	Resource
Function	Describe Current Resource				

Methods	Query String(s)	Inbound Data	Return Result
GET	None	None	<ResourceDescription>
Notes	Returns a description of the resource		

9.4. capabilities

URI	Capabilities	Version	1.0	Type	Resource
Function	Return capabilities of Current Resource				
Methods	Query String(s)	Inbound Data	Return Result		
GET	None	None	Resource-Specific		
Notes	Returns a Capabilities description of the resource				

9.5. /PSIA/profile

URI	/PSIA/profile	Version	2.0	Type	Resource
Function	Returns the identity and functional profile of a PSIA compliant node				
Methods	Query String(s)	Inbound Data	Return Result		
GET	None	None	Resource-Specific		
Notes	(See text below)				

The 'profile' resource is ***required*** for all PSIA Service Model v1.1, and later, compliant devices and systems. This resource contains the following key information used for determining what a node is:

- System Identity fields used for maintaining and managing unique IDs for all components in a system.
- Version identification for the PSIA protocol level of a node.
- A list of all the PSIA specifications supported by a node, along with the accompanying version levels.

Once a node is discovered, the "PSIA/profile" resource should be read to determine 'what' a node is. The schema definition for this resource is provided in the Appendices, specifically Section 12.1.7, below.

The contents of the profile.xsd file are described in the table below. Please reference Section 12.1.7 for the details of the XSD that governs the definitions.

Element Name:Type	**Mandatory / Optional**	**Description**
PsiaProfile	MAND.	Root Element of the schema; contains the following.
PsiaProfile::systemID; GUID	MAND.	Required, modifiable, 128-bit X.667 UUID/GUID (psiaCommonTypes) used to manage the node. Prior to assignment, the node should use its native UUID/GUID (following) until a value is assigned by an authorized management entity.
PsiaProfile::activeID; GUID	MAND.	IEC 9834-8/ITU X.667 128-bit UUID/GUID generated by each node using one of the methods outlined in the IEC/ITU standards. This value is read-only.
PsiaProfile::psiaServiceVersion; float	MAND.	PSIA Service Model spec-version supported by the node; This spec is Version "2.0".
PsiaProfile::primaryPsiaSpec;	MAND.	This required element declares a node or device's

PsiaSpecDecl		primary functionality as defined by a PSIA spec. For example, a camera or encoder would declare “ipmd” (IP Media Device spec) as its primary functional specification for defining its functional characteristics. The spec’s version level is also included in the declaration to aid in interoperability.
PsiaProfile::primaryPsiaSpec:: psiaSpecName	MAND.	Spec name tag indicating the PSIA specification implemented as the primary operational model. Tags are: “ipmd” = IP Media Device spec “racm” = Recording & Content Mgmt spec “videoAnalytics” = Video Analytics spec “cmem” = Common Metadata & Event Model spec “areaCtl” = Area Control spec “csec” = Common Security Model spec “other –PSIA” = other/future PSIA spec. “other-Private” = Mfgr spec Only the following specs are valid as a primary/base spec: ipmd, racm, videoAnalytics, or “other…”. Please note that any tag used that starts with “other…: MUST include a full description in the “nodeDescription” (see below).
PsiaProfile::primaryPsiaSpec:: psiaSpecVersion	MAND.	For each spec tag (above) the pertinent version MUST also be reported.
PsiaProfile::otherSpecList; PsiaSpecDecl	OPT.	IF, nodes support other PSIA specs, they must list them in this list element along with the corresponding spec version level. The list uses the same ‘type’ as the above “primaryPsiaSpec” element.
PsiaProfile::nodeDescription; String	OPT.	This field provides a human friendly description of the node. Please note that if the node has advertised a primary spec tag that starts with “other…”, this field becomes MANDATORY.
PsiaProfile::profileList; PsiaProfileDecl	OPT	If nodes support Operational Profiles, they must list them in this element.

The following HTTP exchange example demonstrates the profile of a PSIA compliant camera that also supports PSIA compliant Video Analytics and event notification (using CMEM). First, the request:

```
GET /PSIA/profile HTTP/1.1
…
```

The example camera device then returns the profile response document:

```
HTTP/1.1 200 OK

Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx    (note: xxx = size of XML block)

<?xml version="1.0" encoding="UTF-8" ?>
<PsiaProfile version="1.1">
     <systemID>3F2504E0-4F89-11D3-9A0C-0305E82C3301</systemID>
     <nativeID>3F2504E0-4F89-11D3-9A0C-0305E82C3301</nativeID>
     <psiaServiceVersion>1.1</psiaServiceVersion>
     <primaryPsiaSpec>
            <psiaSpecName>ipmd</psiaSpecName>
            <psiaSpecVersion>1.1</psiaSpecVersion>
            <psiaSpecProfile>core</psiaSpecProfile>
     </primaryPsiaSpec>
     <otherSpecList>
            <psiaSpecDefn>
                   <psiaSpecName>videoAnalytics</psiaSpecName>
                   <psiaSpecVersion>1.0</psiaSpecVersion>
                   <psiaSpecProfile>basic</psiaSpecProfile>
            </psiaSpecDefn>
            <psiaSpecDefn>
                   <psiaSpecName>cmem</psiaSpecName>
                   <psiaSpecVersion>1.1</psiaSpecVersion>
                   <psiaSpecProfile>core</psiaSpecProfile>
            </psiaSpecDefn>
     </otherSpecList>
     <profileList>
            <psiaProfileDefn>
                   <psiaProfileName>EnterpriseAccessControl</psiaProfileName>
                   <psiaProfileVersion>1.0</psiaProfileVersion>
                   <psiaSpec>areaCtl</psiaSpec>
            </psiaProfileDefn>
     </profileList>
</PsiaProfile>
```

Please note that, in the example above, it may be necessary when implementing multiple PSIA specifications that introduce 'profiles' it may be necessary to add a 'xmlns=urn:psialliance.org' namespace qualifier to the 'PsiaProfile' root element such that PSIA 'profile' names do not conflict in the PSIA namespace. E.g. "<PsiaProfile version=1.1" xmlns="urn:psialliance.org">.

10. Common and Systemic PSIA Services

10.1. Common, Global PSIA Services

Certain protocol definitions, services and functions are global in nature to all PSIA systems and devices. These specifications, like the PSIA Service Model, comprise a suite of foundational protocol and data definitions referenced and leveraged by other PSIA specifications and documents. The current list of these PSIA specifications is outlined below along with a brief explanation and the URI referencing their repository location.

- **PSIA Common Metadata and Event Model (CMEM)**: This specification describes the PSIA architecture for all non-multimedia data exchange. This covers the transport, protocol and data definitions for event and descriptive metadata subscription, transmission, and management. Metadata forms include I/O, Motion Detection, Environmental data, Video Analytics, Access Control, Intrusion, and System information. The CMEM specification is located ***on the PSIA web site document docket***".

- **PSIA Common Security Model (CSEC**): This specification defines the PSIA architecture and design for the systemic management of both network and data security. The specification is located ***on the PSIA web site document docket.***

In the next section of this specification, the common "/PSIA/System" resources are specified. The following tables define the requirement level for each resource and service listed.

10.2. **Resource Requirements**

The following tables provide guidance on what is required for each PSIA topology.

	M-S Internet	**M-S LAN**	**P-P Internet**	**P-P LAN**
Identifiers	UUID required	UUID optional	UUID required	UUID optional
Discovery	Not Required	mDNS	Not Required	mDNS

/PSIA Base Service

While the base /PSIA node is required and intrinsic to all

M-S Internet	**M-S LAN**	**P-P Internet**	**P-P LAN**	**Command**	**GET**	**PUT**	**POST**	**DEL**
				Index	✓			
				Indexr	✓			
				description	✓			
✓	✓	✓	✓	profile (new for v1.1).	✓			

/PSIA/System

PSIA System requirements are defined by topologies.

M-S Internet	M-S LAN	P-P Internet	P-P LAN	Command	GET	PUT	POST	DEL
✓	✓			Reboot		✓		
✓	✓			reboot/time	✓	✓	✓	✓
✓	✓			Update (required for Service Model v2.0, and later nodes) Update/index, Update/description	✓			
✓	✓			Update/executables	✓	✓		
*	*			Update/systemFiles (dependent)	✓	✓		
✓	✓			Update/state	✓	✓		
✓	✓			Update/log	✓			
				configurationData	✓	✓		
*	*			factoryReset ***(required for all embedded devices)***		✓		
				deviceInfo	✓	✓		
				supportReport	✓			
✓	✓	✓	✓	Status	✓			
RW	**RW**	RO	RO	time (RW = Read/Write, RO= Read Only)	✓	✓		
				time/localTime	✓	✓		
				time/timeZone	✓	✓		
				time/ntpServers	✓	✓	✓	✓
				time/ntpServers/<ID>	✓	✓		✓
				Logging	✓	✓		
				logging/messages	✓			

/PSIA/System/Network

M-S Internet	M-S LAN	P-P Internet	P-P LAN	Command	GET	PUT	POST	DEL
✓	✓			Interfaces	✓			
✓	✓			interfaces/<ID>	✓	✓		
✓	✓			interfaces/<ID>/ipAddress	✓	✓		
				interfaces/<ID>/wireless	✓	✓		
				interfaces/<ID>/ieee802.1x	✓	✓		
				interfaces/<ID>/ipFilter	✓	✓		

M-S Internet	M-S LAN	P-P Internet	P-P LAN	Command	GET	PUT	POST	DEL
				interfaces/<ID>/ipFilter/filterAddresses	✓	✓	✓	✓
				interfaces/<ID>/ipFilter/filterAddresses/<ID>	✓	✓		✓
				interfaces/<ID>/snmp	✓	✓		
				interfaces/<ID>/snmp/v2c	✓	✓		
				interfaces/<ID>/snmp/v2c/trapReceivers	✓	✓	✓	✓
				interfaces/<ID>/snmp/v2c/trapReceivers/<ID>	✓	✓		✓
				interfaces/<ID>/snmp/advanced	✓	✓		
				interfaces/<ID>/snmp/advanced/users	✓	✓	✓	✓
				interfaces/<ID>/snmp/advanced/users/<ID>	✓	✓		✓
				interfaces/<ID>/snmp/advanced/notificationFilters	✓	✓	✓	✓
				interfaces/<ID>/snmp/advanced/notificationFilters/<ID>	✓	✓		✓
				interfaces/<ID>/snmp/advanced/notificationReceivers	✓	✓	✓	✓
				interfaces/<ID>/snmp/advanced/notificationReceivers/<ID>	✓	✓		✓
				interfaces/<ID>/snmp/v3	✓	✓		
				interfaces/<ID>/qos	✓	✓		
				interfaces/<ID>/qos/cos	✓	✓	✓	✓
				interfaces/<ID>/qos/cos/<ID>	✓	✓		✓
				interfaces/<ID>/qos/dscp	✓	✓	✓	✓
				interfaces/<ID>/qos/dscp/<ID>	✓	✓		✓
				interfaces/<ID>/discovery	✓	✓		
				interfaces/<ID>/syslog	✓	✓		
				interfaces/<ID>/syslog/servers	✓	✓	✓	✓
				interfaces/<ID>/syslog/servers/<ID>	✓	✓		✓

(Optional or Dependent /PSIA/System Resources)

/PSIA/System/Storage

Command	GET	PUT	POST	DEL
Volumes	✓			
volumes/<ID>	✓			
volumes/<ID>/status	✓			
volumes/<ID>/format		✓		
volumes/<ID>/files	✓			✓
volumes/<ID>/files/<ID>	✓			✓
volumes/<ID>/files/<ID>/data	✓			

/PSIA/System/IO

Command	GET	PUT	POST	DEL
Status	✓			
Inputs	✓			
inputs/<ID>	✓	✓		
inputs/<ID>/status	✓			
Outputs	✓			
outputs/<ID>	✓	✓		
outputs/<ID>/trigger		✓		
outputs/<ID>/status	✓			

/PSIA/System/Audio and /PSIA/System/Video

See IP Media Device specification for resource requirements.

/PSIA/System/Serial

Command	GET	PUT	POST	DEL
Ports	✓			
ports/<ID>	✓	✓		
ports/<ID>/command		✓		

10.3. /PSIA/System : Common System Services

All "/PSIA/System" resources are considered 'Admin'-only level resources and are to be strictly governed by the User permissions, and session level security, defined in the PSIA Common Security Model (CSEC) specification. Many of the functions offered by the 'System' resources are critical, or destructive, or potentially disruptive, to normal operation (e.g. "/PSIA/System/reboot"). All of the resources contained in the 'System' resource hierarchy are administrative and managerial in nature. Therefore, the only instance where these resources are available without administrative permission clearance is during PSIA ordained automated compliance testing.

All PSIA implementations prior to Service Model v2.0 MUST implement the "/PSIA/Security" service defined in the IP Media Device specification v1.1, or an equivalent or greater level of security function.

All unauthenticated, unauthorized transactions to the "/PSIA/System" interfaces defined herein will respond with an HTTP status code of 401 "Unauthorized."

URI	/PSIA/System		**Type**	Service
Function	System services.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
Notes				

9.3.1 /PSIA/System/reboot

URI	/PSIA/System/reboot		**Type**	Resource
Function	Reboot the device.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
PUT			<ResponseStatus>	
Notes	The <ResponseStatus> XML data is returned before the device proceeds to reboot.			

The '/PSIA/System/reboot' resource performs reboot operations when a PUT is issued to it. However, for Service Model v2.0, if an Update process (see Section 9.3.3) is underway, no reboots are allowed until an appropriate state is reached in the Update processes. When a reboot cannot be honored, based on state, an HTTP Status Code of '409 Conflict' is returned.

9.3.1.1 /PSIA/System/reboot/time

URI	/PSIA/System/reboot/time		**Type**	Resource
Function	Reboot a device/system in a scheduled manner			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<BootTime>	
PUT		<BootTime>	<ResponseStatus>	
POST		<BootTime>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	The <ResponseStatus> document is returned in all write/delete cases indicating either success or failure of the operation.			
BootTime.xsd				

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

All Service Model v2.0 compliant PSIA nodes must implement the ability for reboots to be scheduled. All PUTs (updates), POSTs ('BootTime' creation when none exists), and DELETEs are governed by administrative access rights. GETs (reads) are allowed for any entity unless prohibited by user permissions. The '/PSIA/System/reboot/time' resource does not affect the operation of the 'reboot' resource. Additionally, a PUT to the '/PSIA/System/reboot' resource overrides a scheduled reboot. All reboot processes delete any scheduled 'BootTime' (i.e. 'BootTime' does not survive reboots). Just like '/PSIA/System/reboot',(see above) if an Update is in progress, reboots are locked-out until all updates are finished.

9.3.2 /PSIA/System/updateFirmware (deprecated)

URI	/PSIA/System/updateFirmware		Type	Resource
Function	Update the firmware of the device.			
Methods	Query String(s)	Inbound Data	Return Result	
PUT		<opaque executable image/payload>	<ResponseStatus>	
Notes	After successful completion of this API, the <ResponseStatus> XML data is returned, the device closes the HTTP/TCP connection, and the device proceeds to reboot.			

The transfer of executable images should be transferred using HTTP 1.1 'chunking' (see RFC 2616; recommended chunk size of 8KB), and the correct 'Content-type' value.

9.3.3 /PSIA/System/Update

The "/PSIA/System/Update" Service is new for Service Model v2.0. It replaces the older 'updateFirmware' resource with a more complete and manageable set of update-related resources. These resources, described below, enable the updating of executables, system files (if present), and the time at which an update process is to be activated (i.e. 'committed'). Unlike the 'updateFirmware' resource, the reboot process must be explicitly engaged either via the 'commit' resource, or via the "/PSIA/System/reboot" resource listed in ***Section 9.3.1***. As a PSIA Service, the 'Update' resource must have the 'index' and 'description'' resources to identify the subordinate resource structure.

URI	/PSIA/System/Update		Type	Service
Function	Update facility/service for PSIA nodes.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<UpdateService>	
Notes	The Update service allows readers to GET a copy of its 'UpdateService' information. This info includes the version of the Update service, and the resource structure of the service. This enables readers to GET the version, plus the 'index' information in one read operation. The 'UpdateService' XSD is listed below.			

updateService.xsd

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

<xs:element name="updateVersion" minOccurs="1" maxOccurs="1" type="xs:int"/>
<xs:element name="updateActivityTimeout" minOccurs="1" maxOccurs="1" type="xs:int"/>
<xs:element name="updateResourceList" minOccurs="1" maxOccurs="1" type="UpdateResourceInfo"/>

<xs:complexType name="UpdateResourceInfo">
  <xs:sequence>
       <xs:element name="numberOfResources" minOccurs="1" maxOccurs="1" type="xs:int"/>
       <xs:element name="resourceName" minOccurs="1" maxOccurs="unbounded" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:schema>
```

Examples

```
<?xml version="1.0" encoding="UTF-8"?>
<updateVersion>1</updateVersion>
<updateInactivityTimeout>120</updateInactivityTimeout>
<updateResourceList>
        <numberOfResource>2</numberOfResources>
        <resourceName>executables</resourceName>
        <resourceName>state</resourceName>
</updateResourceList>

The above example is for a basic device/system that has the 'executables' and 'state'
resources. This device/system has no update-able system files. The Update Inactivity Timeout is
set to 120 seconds (2 minutes).
```

```
<?xml version="1.0" encoding="UTF-8"?>
<updateVersion>1.0</updateVersion>
<updateResourceList>
        <numberOfResource>2</numberOfResources>
        <resourceName>executables</resourceName>
        <resourceName>systemFiles</resourceName>
        <resourceName>state</resourceName>

</updateResourceList>


This example is for a system/device that has 'executables', 'state' and 'systemFiles'resources.
The Update Inactivity Timeout is set to 300 seconds (5 minutes).
```

The "UpdateService" schema defines 3 primary elements:

- “updateVersion”: This integer defines the version level implemented for a particular Update service instance. The version for this specification level is “1” (one).
- “updateInactivityTimeout”: All nodes MUST implement an inactivity timer to detect lapses in the transfer of update files. This value indicates the timeout threshold in SECONDS. Recommended values should range from 60 to 300 seconds (1 to 5 minutes). This timer is reset/restarted upon the receipt of update data. Once a file transfer is engaged, the timeout should trigger under one of 2 possible conditions:
    - A) The file transfer goes inactive for greater than, or equal to, the timeout threshold; Or.
    - B) For multi-file update systems, an insufficient number of files are updated for there to be a complete set of update files, and the timeout lapses.

  If either of the above conditions occur, the device/system enters the “pendingIncomplete” state (see next resource “…/Update/state”) and awaits further update activity.
- “updateResourceList”: This list identifies the resource objects that comprise each particular implementation of a PSIA Update Service. These elements contain the information also declared in the ‘index’ resource thus saving the interrogator the need to read both resources.

Further description of the Inactivity Timer, plus its management and effects, are covered in the next section.

9.3.3.1 /PSIA/System/Update/state

URI	/PSIA/System/Update/state		Type	Resource
Function	Represents/declares current state of Update process(es)			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<UpdateState>	
PUT	“update=done”		<ResponseStatus>	
Notes	The GET interface allows clients to read the current operational of the Update Service and its process(es).			

updateState.xsd

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

<xs:element name="updateStatus" type="UpdateStateInfo"/>

<xs:complexType name="UpdateStateInfo">
       <xs:sequence>
              <xs:element name="UpdateState" minOccurs="1" maxOccurs="1"
type="UpdateOpStates"/>
              <xs:element name="updateErrorInfo" minOccurs="0" maxOccurs="1"
type="UpdateError"/>
       </xs:sequence>
</xs:complexType>
```

```
<xs:simpleType name="UpdateOpStates">
       <xs:restriction base="xs:string">
              <xs:enumeration value="inactive"/>
              <xs:enumeration value="lockedUpdating"/>
              <xs:enumeration value="pendingRestart"/>
              <xs:enumeration value="pendingIncomplete"/>
              <xs:enumeration value="errored"/>
       </xs:restriction>
</xs:simpleType>

<xs:complexType name="UpdateError">
  <xs:sequence>
    <xs:element name="errorCode" minOccurs="1" maxOccurs="1" type="xs:int"/>
    <xs:element name="errorInfo" minOccurs="0" maxOccurs="1" type="xs:string"/>
    <xs:element name="recommendedActions" minOccurs="0" maxOccurs="1" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

Examples

```
<?xml version="1.0" encoding="UTF-8"?>
<updateStatus>
    <updateState version=”1”>inactive</updateState>
</updateStatus>
```

In this example there is no activity engaged or pending with the Update Service.

```
<?xml version="1.0" encoding="UTF-8"?>
<updateStatus>
   <updateState version=”1”>lockedUpdating</updateState>
</updateStatus>
```

The above example the Update Service has update/file transfers actively occurring.

```
<?xml version="1.0" encoding="UTF-8"?>
<updateStatus>
   <updateState version=”1”>pendingComplete</updateState>
</updateStatus>
```

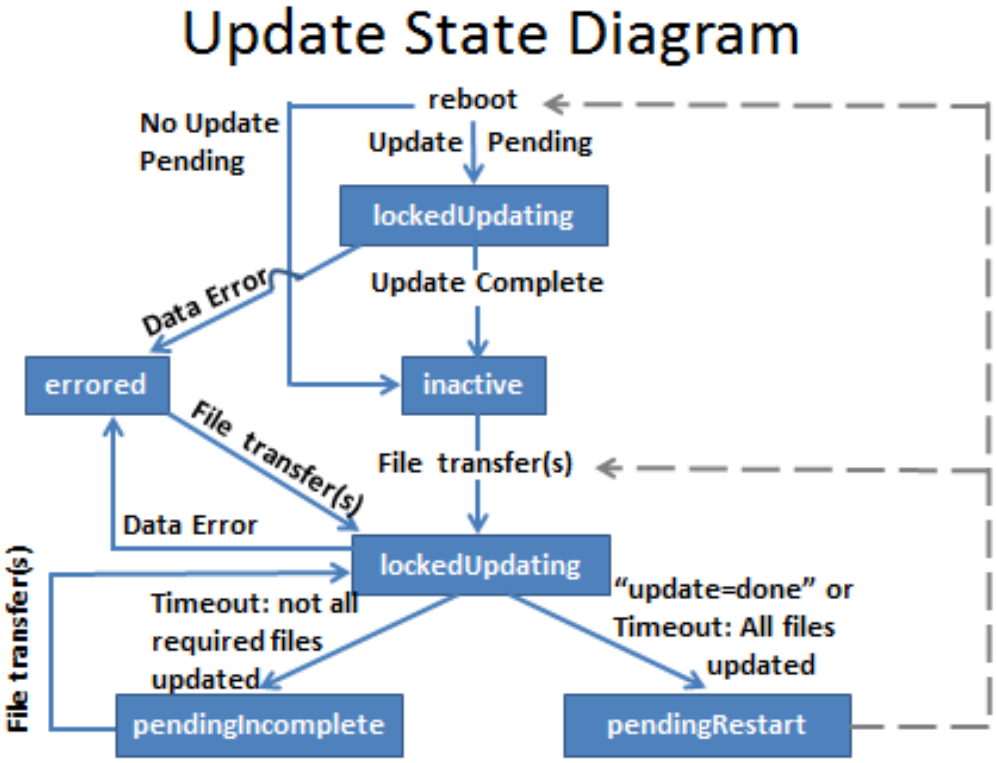
The above example the file transfers related to updating a device/system are complete and the node is waiting for a reboot to occur.

```
<?xml version="1.0" encoding="UTF-8"?>
<updateStatus>
   <UpdateState version=”1”>pendingIncomplete</UpdateState>
</updateStatus>
```

The above example the file transfers related to updating a device/system never completed successfully (based on the inactivity timerout) and the device/system is awaiting a re-engagement of the update transfers.

```
<?xml version="1.0" encoding="UTF-8"?>
<updateStatus>
   <updateState version="1">errored</updateState>
   <updateErrorInfo>
        <errorCode>94</errorCode>
        <errorInfo>Data corruption of update files occurred</errorInfo>
        <recommendedActions>Re-transfer update files again</recommendedActions>
   </updateErrorInfo>
</updateStatus>

The above example is for a device/system that encountered a data corruption issue while
unpacking/decompressing its newly transferred update files. This node gives the full set of
error information: error code, plus error info, plus recommended actions.
```



=

The above state diagram depicts and describes the respective states of the PSIA Update Service and the actions that stimulate transitions from state to state. Devices and Systems either commit updates pre-reboot, or post reboot. Those systems that do their 'updating' completely prior to rebooting do NOT ever manifest the 'updating' state. The 'updating' state is only present on device/systems that go through an 'unpacking' process after a system reboot/restart. The table below described each state:

Update State	***Description***
Inactive	This state indicates that no activity has occurred, or is actively occurring with the

	Update Service since the last system restart. The only way to transition out of this state is to start active file transfer(s) for the required update information, which forces a transition to the 'locked' state.
lockedUpdating	The 'lockedUpdating' state covers all the activities and processes associated with the transfer of update files, *and*, the process of unpacking/decompressing update information files. While this state is active, all other update activities, except reading the state, are locked out. This state indicates the following: A) That active update file transfers are in-progress and all other activities are locked out. And, … B) The files transferred are being prepared for update readiness. Please note that for systems that must initiate some form of 'update' processing after reboot, this state is held until the update processing is complete. For systems that do not require post-boot update processing, this state only occurs during file transfer and prep processing. There are only 3 ways to transition out of the 'lockedUpdating' state: A) All required file transfers are completed, the client performs a PUT to the /PSIA/System/Update/state resource with the "update=done" QSP, AND all update data is ready for activation (i.e. 'unpacked', validated, etc.). This case transitions to the 'pendingComplete' state; Or… B) If an inactivity timeout occurs during file transfers, or the complete set of required files was not transferred before the inactivity timer expired, the 'locked' state transitions to 'pendingIncomplete' state; Or, … C) An unrecoverable error occurs during the 'lockedUpdating' processes forcing a transition to the 'errored' state (see below). NOTE: While in the 'lockedUpdating;' state, ALL 'reboots' are prohibited. Please see Section 9.3.1 for more details.
pendingComplete	This state indicates that a node is ready to be rebooted/restarted. ALL files are fully prepared and ready. The only transitions out of this state are based on the following events: A) A reboot occurs (the expected event); Or… B) Another set of file transfers occurs thus moving the node back to the 'locked' state. This is allowed but NOT recommended. It is only allowed for emergency update purposes.
pendingIncomplete	This state only occurs, during a timeout condition while being updated in the 'lockedUpdating' state. The only way to exit this state is for the management client to re-initiate the update process.
Errored	This state occurs as the result of an unrecoverable, uncorrectable error encountered during the update processes associated with the 'lockedUpdating' activities (see above).

The '/PSIA/System/Update/state' resource enables updating entities to read the current state of a node being update during the various phases of the update process. Please note that the target node, alone, updates its states as it works through the update processes.

9.3.3.2 /PSIA/System/Update/executables

URI	/PSIA/System/Update/executables	Type	Resource

Function	Update the executable image(s) of a system or device.		
Methods	**Query String(s)**	**Inbound Data**	**Return Result**
GET			<ExecutableImageList
PUT	*Optional: "name=<filename>",*	Binary software package	<ResponseStatus>
Notes	The GET interface allows clients to read information on the number, and optionally, the types of executable images that a node needs for a complete update. For nodes that have multiple images, the schema document indicates if the node requires a 'complete' update (all images must be updated), or 'partial' updates (any subset of the images). For single executable image nodes, the value is always 'complete'. Images are transferred to the "/PSIA/System/update/Executables' resource. There is a 1:1 ratio of PUTs to images (i.e. not multi-image transfers with a single PUT). For multi-image systems, the "name=…" QSP *MUST* be used to convey the accompanying image's filename. For single image systems, this QSP is not required, but is recommended. After successful completion of this API, the <ResponseStatus> XML data is returned. The updated unit then waits for the updating node to 'reboot' the unit vial the '/PSIA/System/reboot' resource' (See Section 9.3.1, above).		

executableImages.xsd

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

<xs:complexType name="ExecutableImageList">
       <xs:attribute version="1.0"/>
       <xs:element name="numberOfImages" minCount="1" maxCount="1" type="xs:int"/>
       <xs:element name="updateType" minCount="1" maxCount="1" type="UpdateType"/>
       <xs:element name="imageInfo" minCount="1" maxCount="unbounded" type="FileInfo"/>
</xs:complexType>

<xs:simpleType name="UpdateType">
       <xs:restriction base="xs:string">
              <xs:enumeration value="complete"/>
              <xs:enumeration value="partial"/>
       </xs:restriction>
</xs:simpleType>

<xs:complexType name="FileInfo">
       <xs:annotation>
              <xs:documentation>
              The next element gives the filename of a file. It may contain 'wildcards'
              since many system/exe files have date-codes/tags in their file names.
              The file format element is a MIME identifier for what type of file,
              the file being identified is. Binary files MUST use either "application/octet-
              stream"
              or "application/bin" (for logic/microcode files).
```

```
                </xs:documentation>
        </xs:annotation>
        <xs:element name="fileName" minCount="1" maxCount="1" type="xs:string"/>
        <xs:element name="fileFormat" minCount="0" maxCount="1" type="xs:string"/>
</xs:complexType>

</xs:schema>
```

Examples

```
<?xml version="1.0" encoding="UTF-8"?>
<ExecutableImageList version="1.0">
    <numberOfImages>2</numberOfImages>
    <updateType>partial</updateType>
    <ImageInfo>
        <fileName>bios.rbf</fileName>
        <fileFormat>application/bin</fileFormat>
    </ImageInfo>
    <ImageInfo>
        <fileName >system.bin.tar.gz</fileName>
        <fileFormat>application/octet-stream</fileFormat>
    </ImageInfo>
</ExecutableImageList>

In the above example, the target system/device has 2 executable images that may be updated
("bios.rbf", "system.bin.tar"). This node allows partial updates to be committed.
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ExecutableImageList version="1.0">
    <numberOfImages>1</numberOfImages>
    <updateType>complete</updateType>
</ExecutableImageList>

The above example is for a simple system/device that has a single update-able executable image.
```

9.3.3.3 /PSIA/System/Update/systemFiles

URI	/PSIA/System/Update/systemFiles		Type	Resource
Function	Update the (optional) system files of a device or system.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<SystemFileList>	
PUT	*"name=<filename>",*	File data	<ResponseStatus>	
POST	*"name=<filename>"*	File data	<ResponseStatus>	

Notes	This, optional, Update resource is employed by PSIA nodes that have 'system' files that are part of the core, or key applications', operation. A 'GET' returns a document that details what update-able files the device/system retains. A 'PUT' MUST be used to update/replace existing files, and a 'POST' MUST be used to create new files. Please note that the node MUST indicate if partial, or complete, updates are allowed.

systemFiles.xsd

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

<xs:include schemaLocation="executableImages.xsd"/>

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

<xs:complexType name="SystemFileList">
  <xs:sequence>
    <xs:attribute version="1.0"/>
    <xs:element name="updateType" minCount="0" maxCount="1" type="UpdateType"/>
    <xs:element name="numberOfFiles" minCount="1" maxCount="1" type="xs:int"/>
    <xs:element name="updateType" minCount="1" maxCount="1" type="UpdateType"/>
    <xs:element name="systemFileInfo" minCount="1" maxCount="unbounded" type="FileInfo"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Examples

```
<?xml version="1.0" encoding="UTF-8"?>
<SystemFileList version="1.0">
    <updateType>partial</updateType>
    <numberOfFiles>2</numberOfFiles>
    <systemFileInfo>
        <fileName>system.dat</fileName>
        <fileFormat>application/bin</fileFormat>
    </systemFileInfo>
    <systemFileInfo>
        <fileName>config.info</fileName>
        <fileFormat>application/text</fileFormat>
    </systemFileInfo>
</SystemFileImageList>

In the above example, the target system/device has 2 system files listed. The first,
"system.dat" is a binary file. The second, "config.info", is a text file. This document
indicates that the device/system allows partial updates.
```

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

```
    <updateType>complete</updateType>
    <numberOfImages>1</numberOfImages>
    <systemFileInfo>
        <fileName>system.dat</fileName>
        <fileFormat>application/bin</fileFormat>
    </systemFileInfo>
</SystemFileList>

The above example is for a simple system/device that has a single update-able executable image.
```

9.3.3.4 /PSIA/System/Update/log

URI	/PSIA/System/Update/log		Type	Resource
Function	Provides an Update activity history			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<UpdateLog>	
Notes	The Update log is a read-only history of the last 'n' number of updates. ALL nodes are required to keep a history of at least 2 prior updates.			

UpdateLog.xsd

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

<xs:complexType name="UpdateHistory">
       <xs:sequence>
              <xs:element name="updateLogCount" minCount="1" maxCount="1" type="xs:int"/>
              <xs:element name="maxEntries" minCount="1" maxCount="1" type="xs:int"/>
              <xs:element name="updateList" minCount="1" maxCount="unbounded"
                     type="UpdateList"/>
       </xs:sequence>
</xs:complexType>

<xs:simpleType name="UpdateList">
       <xs:element name="updateInfo" minCount="1" maxCount="unbounded" type="UpdateInfo"/>
</xs:simpleType>

<xs:complexType name="UpdateInfo">
       <xs:sequence>
              <xs:element name="updateTime" minCount="1" maxCount="1" type="xs:dateTime"/>
              <xs:element name="updateVersion" minCount="1" maxCount="1" type="xs:string"/>
              <xs:element name="updateStatus" minCount="1" maxCount="1" type="UpdateStatus"/>
              <xs:element name="updateExecFileCount" minCount="1" maxCount="1"
                     type="xs:dateTime"/>
```

```
                <xs:annotation>
                        <xs:documentation>
                                The following element is required for all nodes that
                                have updateable system files. Otherwise it is
                                unnecessary.
                        </xs:documentation>
                </xs:annotation>
                <xs:element name="updateSysFileCount" minCount="0" maxCount="1"
                        type="xs:dateTime"/>
        </xs:sequence>
</xs:complexType>

<xs:simpleType name="UpdateStatus">
        <xs:restriction base="xs:string">
                <xs:enumeration value="ok"/>
                <xs:enumeration value="incomplete"/>
                <xs:enumeration value="failed"/>
        </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Examples

```
<?xml version="1.0" encoding="UTF-8"?>
<UpdateLog>
        <updateLogCount>1</updateLogCount>
        <maxEntries>4</maxEntries>
        <updateList>
                <updateInfo>
                        <updateTime>20011-05-30T021:30:44Z</updateTime>
                        <updateVersion>1.1</updateVersion>
                        <updateStatus>ok</updateStatus>
                        <updateExecFileCount>2</updateExecFileCount>
                        <updateSysFileCount>1</updateSysFileCount>
                </updateInfo>
                <updateInfo>
                        <updateTime>20012-01-27T023:16:10Z</updateTime>
                        <updateVersion>1.2</updateVersion>
                        <updateStatus>ok</updateStatus>
                        <updateExecFileCount>2</updateExecFileCount>
                        <updateSysFileCount>1</updateSysFileCount>
                </updateInfo>
        </updateList>
</UpdateLog>

This example is for a node that has had 2 udpates. One for v1.1, and another for v1.2. Each
update process was comprised of 3 files each.
```

The '/PSIA/System/Update/log' resource is optional, but highly recommended for all PSIA nodes implementing the Update Service. This resource object maintains a list of the last 'n' updates. The

minimum number of log entries is 2. The 'updateVersion' element is a string to allow for various version formats.

9.3.3.5 Notes On Updating

Use of the Update Service is relatively simple. A Management client reads the Update resources to determine what files need to be transferred. Upon the commencement of data transfer, the node being updated locks out other 'updaters' and reboot operations (see Section 9.3.1) while updating its states as the update processes progress. The updating node performs a PUT to the '/PSIA/Update/state' resource object with the "update=done" QSP to signal that it considers the update complete from the client's side. This PUT triggers the updated node to validate its files (if that has not already been performed) and render a resulting update state. 'PendingComplete' indicates that all items are OK and the updated node is awaiting a reboot. The 'pendingIncomplete' and 'errored' states indicate that the update process did not successfully complete. In each of these cases, the management client must reinitiate the update process and/or log its own update error event.

Nodes being updated are responsible for monitoring update activity via an inactivity timer. A recommended inactivity timeout value of 30 seconds should be employed unless conditions prohibit. A timeout can only result in one of 2 possible states once an update process has begun: A) 'pendingIncomplete' is imposed when a timeout occurs without the required file set being completely transferred; or B), 'pendingRestart' if all necessary files have been successfully transferred. Update clients MUST read the update state in order to determine how to complete an update should a timeout occur.

Though the implementation of the '/PSIA/System/Update/log' resource is not required, it is highly recommended for implementation. It keeps an update history with each node irrespective of each node's deployment, configuration, or movement from system to system.

Finally, the implementation of devices/systems that expose multi-file update 'directories' introduce a level of complexity regarding the determination of when an update transfer is complete, how recoveries should be performed in the case of errors, and increase the number of object transfers. We encourage all developers to keep things as simple as possible. In all implementations, developers should exercise great care in ensuring that error recovery cases are all covered such that no node can be left stranded inoperable (unless due to power outage, etc.).

9.3.4 /PSIA/System/configurationData

URI	`/PSIA/System/configurationData`		**Type**	Resource
Function	The function is used to get or set the configuration data for the device. This is opaque data that can be used to save and restore the device configuration.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			Opaque data	
PUT		Opaque Data	<ResponseStatus>	
Notes	Configuration data is device-dependent – it may be binary or any other format. Client may use the HTTP `Accept:` header field to inform server what formats are expected. The action of updating a node's configuration data, 'en masse', may cause a node to either A) require a client initiated reboot, B) cause the node to reboot on its own, or C) cause no resetting at all. If a node needs a client to reboot it, the statusCode in the response status must be "7" (Reboot required). If a node is going to reboot on its own, the statusCode must be "2" (Device busy). Otherwise all other operations should render			

	the appropriate statusCode.

The transfer of configuration data greater than 8 KB in size should be transferred using HTTP 1.1 'chunking' (see RFC 2616; recommended chunk size of 8KB), and the correct 'Content-type' value.

9.3.5 /PSIA/System/factoryReset

URI	/PSIA/System/factoryReset		Type	Resource
Function	This function is used to reset the configuration for the device to the factory default.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
PUT	Mode		<ResponseStatus>	
Notes	Two factory reset modes are supported: "full" resets all device parameters and settings to their factory values. "basic" resets all device parameters and settings except the values in /PSIA/System/Network and /PSIA/CSEC. The default mode is "full". The action of resetting a node's entire parameter base may cause a node to either A) require a client initiated reboot, B) cause the node to reboot on its own, or C) cause no rebooting action at all. If a node needs a client to reboot it, the statusCode in the response status must be "7" (Reboot required). If a node is going to reboot on its own, the statusCode must be "2" (Device busy). Otherwise all other operations should render the appropriate statusCode.			

9.3.6 /PSIA/System/deviceInfo

URI	/PSIA/System/deviceInfo		Type	Resource
Function	This function is used to get or set device information.			
Methods	**Query String(s)**	**Data**	**Return Result**	
GET			<DeviceInfo>	
PUT		<DeviceInfo>	<ResponseStatus>	
Notes	Some fields are read-only and may not be set. If these fields are present in the inbound XML block, they are ignored. For the <DeviceInfo> uploaded to the device during a PUT operation, all fields are considered optional and any fields that are not present in the inbound XML are not changed on the device. This allows setting of the fields individually without having to load the entire XML block to the device. <deviceDescription> is a description of the device as defined in RFC1213. <deviceLocation> is the location of the device as defined in RFC1213 <systemContact> is the contact information for the device as defined in RFC1213.			

	<systemObjectID> is the System Object Identifier defined in RFC1213 is required by RFC 1213 and is present, but immutable (not subject to edits.

DeviceInfo XML Block

```
<DeviceInfo version="1.0" xmlns="urn:psialliance-org">
      <deviceName>                <!-- req, xs:string -->                    </deviceName>
      <deviceID>                        <!-- ro, req, xs:string;uuid -->     </deviceID>
      <deviceDescription>   <!-- opt, xs:string -->                    </deviceDescription>
      <deviceLocation>            <!-- opt, xs:string -->
      </deviceLocation>
      <systemContact>             <!-- opt, xs:string -->
      </systemContact>
            <!-- Note: The following are read-only parameters -->
      <model>                           <!-- ro, req, xs:string -->          </model>
      <serialNumber>              <!-- ro, req, xs:string -->          </serialNumber>
      <macAddress>                <!-- ro, req, xs:string;       -->    </macAddress>
      <firmwareVersion>           <!-- ro, req, xs:string -->          </firmwareVersion>
      <firmwareReleasedDate><!-- ro, opt, xs:string -->          </firmwareReleasedDate>
      <logicVersion>              <!-- ro, opt, xs:string -->          </logicVersion>
      <logicReleasedDate>   <!-- ro, opt, xs:string -->          </logicReleasedDate>
      <bootVersion>               <!-- ro, opt, xs:string -->          </bootVersion>
      <bootReleasedDate>          <!-- ro, opt, xs:string -->          </bootReleasedDate>
      <rescueVersion>             <!-- ro, opt, xs:string -->          </rescueVersion>
      <rescueReleasedDate>  <!-- ro, opt, xs:string -->          </rescueReleasedDate>
      <hardwareVersion>           <!-- ro, opt, xs:string -->          </hardwareVersion>
      <systemObjectID>            <!-- ro, opt, xs:string -->          </systemObjectID>
</DeviceInfo>
```

9.3.7 /PSIA/System/supportReport

URI	/PSIA/System/supportReport		Type	Resource
Function	This function is used to get a compressed archive of support information for the device. The archive must contain at least the device's current configuration and log files. Other items that might also be packaged include syslog and operating system information, statistics, etc.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			Support Data	
Notes	The format of the archive is device-dependent (could be tar, zip, etc.). Use http `Accept:` header field to inform server what formats are accepted by client.			

9.3.8 /PSIA/System/status

URI	/PSIA/System/status	Type	Resource

Function	This function is used to get the status of the device.		
Methods	**Query String(s)**	**Inbound Data**	**Return Result**
GET			<DeviceStatus>
Notes	Not all fields of <DeviceStatus> may be present.		

DeviceStatus XML Block

```
<DeviceStatus version="1.0" xmlns="urn:psialliance-org">
      <currentDeviceTime>   <!-- opt, xs:datetime -->                  </currentDeviceTime>
      <deviceUpTime>               <!-- opt, xs:integer, seconds -->    </deviceUpTime>
      <TemperatureList>            <!-- opt -->
            <Temperature>
                  <tempSensorDescription>       <!-- req, xs:string -->
      </tempSensorDescription>
                  <temperature>                         <!-- req, xs:float -->
      </temperature>
            </Temperature>
      </TemperatureList>
      <FanList>                           <!-- opt -->
            <Fan>
                  <fanDescription>                    <!-- req, xs:string -->
      </fanDescription>
                  <speed>                                     <!-- req, xs:integer -->
      </speed>
            </Fan>
      </FanList>
      <PressureList>               <!-- opt -->
            <Pressure>
                  <pressureSensorDescription> <!-- req, xs:string --
></pressureSensorDescription>
                  <pressure>            <!-- req, xs:integer -->      </pressure>
            </Pressure>
      </PressureList>
      <TamperList>                 <!-- opt -->
            <Tamper>
                  <tamperSensorDescription>            <!-- req, xs:string -->
      </tamperSensorDescription>
                  <tamper>                                    <!-- req, xs:boolean -->
      </tamper>
            </Tamper>
      </TamperList>
      <CPUList>                           <!-- opt -->
            <CPU>
                  <cpuDescription>      <!-- req, xs:string -->
            </cpuDescription>
                  <cpuUtilization>      <!-- req, xs:integer, percentage 0..100 -->
      </cpuUtilization>
            </CPU>
      </CPUList>
```

```
      <MemoryList>                  <!-- opt -->
            <Memory>
                  <memoryDescription>    <!-- req, xs:string -->
      </memoryDescription>
                  <memoryUsage>                <!-- req, xs:float, in MB --> </memoryUsage>
                  <memoryAvailable>            <!-- req, xs:float, in MB-->
      </memoryAvailable>
            </Memory>
      </MemoryList>
      <openFileHandles>             <!-- opt, xs:integer -->      </openFileHandles>
</DeviceStatus>
```

9.3.9 /PSIA/System/time

URI	/PSIA/System/time		Type	Resource
Function	Access the device time information.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<Time>	
PUT	timeMode localTime timeZone (Conditional, see below_	<Time>	<ResponseStatus>	
Notes	The QSPs are provided as lightweight parameters for updates. If QSPs are present, then the HTTP PUT **must NOT** have a payload (i.e. <Time> If <timeMode> is set to “manual” the <localTime> and <timeZone> fields are required. The <LocalTime> block sets the device time. If <timeMode> is set to “NTP”, only the <timeZone> field is required. The device time is set by synchronizing with NTP.			

Time XML Block

```
<Time version="1.0" xmlns="urn:psialliance-org">
      <timeMode>            <!-- req, xs:string, "NTP,manual" -->        </timeMode>
      <localTime>           <!-- req, xs:datetime -->
      </localTime>
      <timeZone>            <!-- req, xs:string, POSIX time zone string; see below -->
      </timeZone>
</Time>
```

9.3.10 /PSIA/System/time/localTime

URI	/PSIA/System/time/localTime		Type	Resource
Function	Access the device local time information.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			ISO 8601 Date-Time String	
PUT		ISO 8601 Date-Time String	<ResponseStatus>	
Notes	An ISO 8601 Date/Time string is accepted and returned. If the date/time value has a time zone, the time is converted into the device's local time zone. If the device time mode is set to "NTP", setting this value has no effect.			

9.3.11 /PSIA/System/time/timeZone

URI	/PSIA/System/time/timeZone		Type	Resource
Function	Access the device time zone.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			Time zone string	
PUT		Time zone string	<ResponseStatus>	

Notes	Time zones are defined by POSIX 1003.1 section 8.3 time zone notations. Note that the value following the +/- is the amount of time that must be *added* to the local time to result in UTC.
	
	Example:
	
	`EST+5EDT01:00:00,M3.2.0/02:00:00,M11.1.0/02:00:00`
	
	Defines eastern standard time as "EST" with a GMT-5 offset. Daylight savings time is called "EDT", is one hour later and begins on the second Sunday of March at 2am and ends on the first Sunday of November at 2am.
	
	`CET-1CEST01:00:00,M3.5.0/02:00:00,M10.5.0/03:00:00`
	
	Defines central European time as GMT+1 with a one-hour daylight savings time ("CEST") that starts on the last Sunday in March at 2am and ends on the last Sunday in October at 3am.			

9.3.12 /PSIA/System/time/ntpServers

URI	/PSIA/System/time/ntpServers		Type	Resource
Function	Access the NTP servers configured for the device.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<NTPServerList>	
PUT		<NTPServerList>	<ResponseStatus>	
POST		<NTPServer>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	When the <timeMode> is set to "NTP", the servers in this list are used to synchronize the device's system time.			

NTPServerList XML Block

```
<NTPServerList version="1.0" xmlns="urn:psialliance-org">
      <NTPServer/>   <!-- opt -->
</NTPServerList>
```

9.3.13 /PSIA/System/time/ntpServers/<ID>

URI	/PSIA/System/time/ntpServers/*ID*		Type	Resource
Function	Access an NTP server configured for the device.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<NTPServer>	
PUT		<NTPServer>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	Depending on the value of <addressingFormatType>, either the <hostName> or the IP address fields will be used to locate the NTP server. Use of IPv4 or IPv6 addresses depends on the value of the <ipVersion> field in /PSIA/System/Network/interfaces/*ID*/ipAddress.			

NTPServer XML Block

```
<NTPServer version="1.0" xmlns="urn:psialliance-org">
      <id>                        <!-- req, xs:string; id -->   </id>
      <addressingFormatType>
            <!-- req, xs:string, "ipaddress,hostname" -->
      </addressingFormatType>
      <hostName>                  <!-- dep, xs:string -->              </hostName>
      <ipAddress>                 <!-- dep, xs:string -->              </ipAddress>
      <ipv6Address>         <!-- dep, xs:string -->              </ipv6Address>
      <portNo>                    <!-- opt, xs:integer -->             </portNo>
</NTPServer>
```

9.4 /PSIA/System/logging

URI	/PSIA/System/logging		Type	Resource
Function	This function is used to access the logging parameters.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<Logging>	
PUT		<Logging>	<ResponseStatus>	
Notes	The device maintains a rolling log of <maxEntries> that can be configured and queried.			

Logging XML Block

```
<Logging version="1.0" xmlns="urn:psialliance-org">
      <LogTrigger>          <!-- opt -->
            <severity>            <!-- req, xs:string, Severities are defined in RFC3164 -->
      </severity>
      </LogTrigger>
      <LocalLog>                  <!-- opt -->
            <maxEntries>   <!-- req, xs:integer -->      </maxEntries>
      </LocalLog>
</Logging>
```

9.4.3 /PSIA/System/logging/messages

URI	/PSIA/System/logging/messages		Type	Resource
Function	This function is used to access the message log.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<LogMessageList>	
Notes	Devices may define additional logging fields for extended information. The message log is read-only.			

LogMessageList XML Block

```
<LogMessageList version="1.0" xmlns="urn:psialliance-org">
      <LogMessage>          <!-- opt -->
            <logNo>               <!-- req, xs:integer -->
            </logNo>
            <dateTime>            <!-- req, xs:datetime -->
            </dateTime>
            <severity>            <!-- req, xs:integer, defined in RFC3164 -->
      </severity>
            <eventID>             <!-- opt, xs:string;id -->
            </eventID>
            <message>             <!-- req, xs:string -->
            </message>
      </LogMessage>
</LogMessageList>
```

9.5 /PSIA/System/Storage

URI	/PSIA/System/Storage		Type	Service
Function	This function is used to access storage parameters.			
Methods	Query String(s)	Inbound Data	Return Result	
Notes	Storage service.			

9.5.3 /PSIA/System/Storage/volumes

URI	/PSIA/System/Storage/volumes		Type	Resource
Function	This function is used to access the storage volumes and files on a device.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<StorageVolumeList>	
Notes	Storage is organized into volumes. Each volume is an individual storage space. Creation and configuration of volumes is outside the scope of this interface, thus the information is available on a read-only basis.			

StorageVolumeList XML Block

```
<StorageVolumeList version="1.0" xmlns="urn:psialliance-org">
      <StorageVolume/>      <!-- ro, opt -->
</StorageVolumeList>
```

9.5.4 /PSIA/System/Storage/volumes/<ID>

URI	/PSIA/System/Storage/volumes/*ID*		Type	Resource
Function	This function is used to access a particular storage volume by its ID.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<StorageVolume>	
Notes	Volume information can only be read using this interface.			

StorageVolume XML Block

```
<StorageVolume version="1.0" xmlns="urn:psialliance-org">
      <id>                              <!-- ro, req, xs:string;id -->       </id>
      <volumeName>                  <!-- ro, req, xs:string -->          </volumeName>
      <volumePath>                  <!-- ro, opt, xs:string -->          </volumePath>
      <volumeDescription>   <!-- ro, opt, xs:string -->          </volumeDescription>
      <volumeType>
            <!-- ro, req, xs:string, "VirtualDisk,RAID0,RAID1,RAID0+1,RAID5", etc -->
      </volumeType>
      <storageDescription>
            <!-- ro, opt, xs:string, "DAS","DAS/USB", etc -->
      </storageDescription>
```

```
      <storageLocation>
            <!-- ro, opt, xs:string, "HDD","Flash","SDIO", etc-->
      </storageLocation>
      <storageType>
            <!-- ro, opt, xs:string, "internal,external" -->
      </storageType>
      <capacity>          <!-- ro, req, xs:float, in MB -->                  </capacity>
</StorageVolume>
```

9.5.5 /PSIA/System/Storage/volumes/<ID>/status

URI	/PSIA/System/Storage/volumes/*ID*/status		Type	Resource
Function	This function is used to query the status of a particular storage.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<StorageVolumeStatus>	
Notes	Query the volume status. Currently only the amount of free space is returned. Devices may extend the XML to allow for querying additional information.			

StorageVolumeStatus XML Block

```
<StorageVolumeStatus version="1.0" xmlns="urn:psialliance-org">
      <freeSpace>          <!-- ro, req, xs:float, in MB -->          </freeSpace>
</StorageVolumeStatus>
```

9.5.6 /PSIA/System/Storage/volumes/<ID>/format

URI	/PSIA/System/Storage/volumes/*ID*/format		Type	Resource
Function	Format a storage volume.			
Methods	Query String(s)	Inbound Data	Return Result	
PUT			<ResponseStatus>	
Notes	Formatting may take time.			

9.5.7 /PSIA/System/Storage/volumes/<ID>/files

URI	/PSIA/System/Storage/volumes/*ID*/files		Type	Resource
Function	Get the list of files stored on a particular storage volume.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<StorageFileList>	
DELETE			<ResponseStatus>	
Notes	Storage files are read-only, except for the possibility to delete. DELETE removes all of the files on the storage volume.			

StorageFileList XML Block

```
<StorageFileList version="1.0" xmlns="urn:psialliance-org">
      <StorageFile/>        <!-- ro, opt -->
</StorageFileList>
```

9.5.8 /PSIA/System/Storage/volumes/<ID>/files/<ID>

URI	/PSIA/System/Storage/volumes/*ID*/files/*ID*		Type	Resource
Function	Access and manipulate a file.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<StorageFile>	
DELETE			<ResponseStatus>	
Notes	DELETE removes a particular file from the storage volume.			

StorageFile XML Block

```
<StorageFile version="1.0" xmlns="urn:psialliance-org">
      <id>                        <!-- ro, req, xs:string;id -->              </id>
      <fileName>                  <!-- ro, req, xs:string -->                 </fileName>
      <fileTimeStamp>       <!-- ro, req, xs:datetime -->               </fileTimeStamp>
      <fileSize>                  <!-- ro, req, xs:float, in MB -->           </fileSize>
</StorageFile>
```

9.5.9 /PSIA/System/Storage/volumes/<ID>/files/<ID>/data

URI	/PSIA/System/Storage/volumes/*ID*/data		Type	Resource
Function	This function is used to access the data of a particular file.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			Raw File Data	
Notes	The video/audio data may be encrypted according to device-dependent specifications. The video/audio format is dependent on device capabilities and configurations. The client may use the HTTP `Accept:` header to negotiate the data format.			

9.6 /PSIA/System/Network

URI	/PSIA/System/Network		Type	Service
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
Notes	System network configuration.			

9.6.3 /PSIA/System/Network/interfaces

URI	/PSIA/System/Network/interfaces		Type	Resource
Function	Access the device network interfaces.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<NetworkInterfaceList>	
Notes	As hardwired system resources, network interfaces cannot be created or destroyed.			

NetworkInterfaceList XML Block

```
<NetworkInterfaceList version="1.0" xmlns="urn:psialliance-org">
      <NetworkInterface/>   <!-- req -->
</NetworkInterfaceList>
```

9.6.4 /PSIA/System/Network/interfaces/<ID>

URI	/PSIA/System/Network/interfaces/*ID*		Type	Resource
Function	Access a particular network interface.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<NetworkInterface>	
PUT		<NetworkInterface>	<ResponseStatus>	
Notes				

NetworkInterface XML Block

```
<NetworkInterface version="1.0" xmlns="urn:psialliance-org">
      <id>                  <!-- ro, req, xs:string;id -->              </id>
      <IPAddress/>   <!-- req -->
      <Wireless/>           <!-- opt -->
      <IEEE802_1x/>  <!-- opt -->
      <IPFilter/>           <!-- opt -->
      <SNMP/>               <!-- opt -->
      <QoS/>                <!-- opt -->
      <Discovery/>   <!-- opt -->
      <Syslog/>             <!-- opt -->
</NetworkInterface>
```

9.6.5 /PSIA/System/Network/interfaces/<ID>/ipAddress

URI	/PSIA/System/Network/interfaces/*ID*/ipAddress		Type	Resource
Function	Access IP addressing settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<IPAddress>	
PUT		<IPAddress>	<ResponseStatus>	
Notes	If <addressingType> is dynamic, fields below it need not be provided. If <addressingType> is dynamic, a DHCP client is used for the device. If <addressingType> is static the device IP address is configured manually and the gateway and DNS fields are optional. If <addressingType> refers to APIPA, the device IP address is automatically configured without DHCP. In this case the gateway and DNS fields are optional. Use of <ipAddress> or <ipv6Address> in fields is dictated by the <ipVersion> field. If <ipVersion> is “v4” the <ipAddress> fields are used; if <ipVersion> is “v6” the <ipv6Address> fields are used. If <ipVersion> is "dual", both <ipAddress> and <ipv6Address> fields may be used. <subnetMask> notation is “xxx.xxx.xxx.xxx”. <IPV6Address> is “xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx” using CIDR notation.			

IPAddress XML Block

```
<IPAddress version="1.0" xmlns="urn:psialliance-org">
      <ipVersion>                  <!-- req, xs:string, "v4,v6,dual" --> </ipVersion>
      <addressingType>       <!-- req, xs:string, "static,dynamic,apipa" -->
      </addressingType>
      <ipAddress>                  <!-- dep, xs:string -->
            </ipAddress>
      <subnetMask>           <!-- dep, xs:string, subnet mask for IPv4 address -->
      </subnetMask>
      <ipv6Address>          <!-- dep, xs:string -->
      </ipv6Address>
      <bitMask>                    <!-- dep, xs:integer,  bitmask IPv6 address -->
      </bitMask>
      <DefaultGateway>       <!-- dep -->
            <ipAddress>            <!-- dep, xs:string -->              </ipAddress>
            <ipv6Address>  <!-- dep, xs:string -->              </ipv6Address>
      </DefaultGateway>
      <PrimaryDNS>           <!-- dep -->
            <ipAddress>            <!-- dep, xs:string -->              </ipAddress>
            <ipv6Address>  <!-- dep, xs:string -->              </ipv6Address>
      </PrimaryDNS>
      <SecondaryDNS>         <!-- dep -->
            <ipAddress>            <!-- dep, xs:string -->              </ipAddress>
            <ipv6Address>  <!-- dep, xs:string -->              </ipv6Address>
      </SecondaryDNS>
</IPAddress>
```

9.6.6 /PSIA/System/Network/interfaces/<ID>/wireless

URI	/PSIA/System/Network/interfaces/*ID*/wireless		Type	Resource
Function	Access wireless network settings.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<Wireless>	
PUT		<Wireless>	<ResponseStatus>	
Notes	If the <securityMode> field is "WEP", the <WEP> block must be provided. If the <securityMode> field is "WPA" or "WPA2-personal", the <WPA> block must be provided. If the "WPA" or "WPA2-enterprise" security mode is used, the <WPA> block must be used and settings related to 802.1x must be set using the /PSIA/System/Network/interfaces/*ID*/ieee802.1x resource. <channel> corresponds to an 802.11g wireless channel number or "auto" for autoconfiguration. <wmmEnabled> enables 802.11e, QoS for IEEE 802.11 networks (Wi-Fi Multimedia) <defaultTransmitKeyIndex> indicates which encryption key is used for WEP security. <encryptionKey> is the WEP encryption key in hexadecimal format. <sharedKey> is the pre-shared key used in WPA			

Wireless XML Block

```
<Wireless version="1.0" xmlns="urn:psialliance-org">
       <enabled>                          <!-- req, xs:boolean -->
       </enabled>
       <wirelessNetworkMode>
              <!-- opt, xs:string, "infrastructure,adhoc" -->
       </wirelessNetworkMode>
       <channel>                          <!-- opt, xs:string, "1-14,auto" -->
       </channel>
       <ssid>                             <!-- opt, xs:string -->              </ssid>
       <wmmEnabled>                  <!-- opt, xs:boolean -->      </wmmEnabled>
       <WirelessSecurity>    <!-- opt -->
              <securityMode>
                     <!-- opt, xs:string,
                     "disable,WEP,WPA-personal,WPA2-personal,WPA-RADIUS,WPA-enterprise,WPA2-
enterprise"
                     -->
              </securityMode>
              <WEP>                      <!-- dep, depends on <securityMode> -->
                     <authenticationType>
                            <!-- req, xs:string, "open,sharedkey,auto" -->
                     </authenticationType>
                     <defaultTransmitKeyIndex>            <!-- req, xs:integer -->
       </defaultTransmitKeyIndex>
                     <wepKeyLength>               <!-- opt, xs:integer "64,128" -->
</wepKeyLength>
                     <EncryptionKeyList>
```

```
                        <encryptionKey>
                                <!-- req, xs:hexBinary, WEP encryption key in hexadecimal
format -->
                        </encryptionKey>
                </EncryptionKeyList>
        </WEP>
        <WPA>                                   <!-- dep, depends on <securityMode> -->
                <algorithmType>       <!-- req, xs:string, "TKIP,AES,TKIP/AES"-->
    </algorithmType>
                <sharedKey>                     <!-- req, xs:string, pre-shared key used in
WPA --> </sharedKey>
        </WPA>
    </WirelessSecurity>
</Wireless>
```

9.6.7 /PSIA/System/Network/interfaces/<ID>/ieee802.1x

URI	/PSIA/System/Network/interfaces/*ID*/ieee802.1x		Type	Resource
Function	Access IEEE 802.1x settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<IEEE802_1x>	
PUT		<IEEE802_1x>	<ResponseStatus>	
Notes	If the <authenticatonProtocolType> tag corresponds to "EAP-TTLS", then the <innerTTLSAuthenticationMethod> tag must be provided. If the <authenticationProtocolType> corresponds to "EAP-PEAP" or "EAP-FAST", then the <innerEAPProtocolType> tag must be provided. The <anonymousID> tag is optional. If the <authenticationProtocolType> corresponds to "EAP-FAST", then the <autoPACProvisioningEnabled> tag must be provided. <anonymousID> is the optional anonymous ID to be used in place of the <userName>.			

IEEE802_1x XML Block

```
<IEEE802_1x version="1.0" xmlns="urn:psialliance-org">
      <enabled>      <!-- req, xs:boolean -->      </enabled>
      <authenticationProtocolType>
            <!-- req, xs:string, "EAP-TLS,EAP-TTLS,EAP-PEAP,EAP-LEAP,EAP-FAST" -->
      </authenticationProtocolType>
      <innerTTLSAuthenticationMethod>
            <!-- dep, xs:string, "MS-CHAP,MS-CHAPv2,PAP,EAP-MD5" -->
      </innerTTLSAuthenticationMethod>
      <innerEAPProtocolType>
            <!-- dep, xs:string, "EAP-POTP,MS-CHAPv2" -->
      </innerEAPProtocolType>
      <validateServerEnabled>       <!-- dep, xs:boolean -->      </validateServerEnabled>
      <userName>                    <!-- dep, xs:string -->       </userName>
      <password>                    <!-- dep, xs:string -->       </password>
      <anonymousID>         <!-- opt, xs:string -->       </anonymousID>
```

```
      <autoPACProvisioningEnabled>  <!-- dep, xs:boolean -->      </autoPACProvisioningEnabled>
</IEEE802_1x>
```

9.5.6 /PSIA/System/Network/interfaces/<ID>/ipFilter

URI	/PSIA/System/Network/interfaces/*ID*/ipFilter		Type	Resource
Function	Access IP filtering settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<IPFilter>	
PUT		<IPFilter>	<ResponseStatus>	
Notes	The <permissionType> field, if provided as a direct child of <IPFilter>, acts as a system level configuration and will apply to all of the <IPFilterAddress> entries, overriding the value provided in a particular <IPFilterAddress> block.			

IPFilter XML Block

```
<IPFilter version="1.0" xmlns="urn:psialliance-org">
      <enabled>                    <!-- req, xs:boolean -->
      </enabled>
      <permissionType>      <!-- opt, xs:string, "deny,allow" -->        </permissionType>
      <IPFilterAddressList/> <!-- opt -->
</IPFilter>
```

9.5.7 /PSIA/System/Network/interfaces/<ID>/ipFilter/filterAddresses

URI	/PSIA/System/Network/interfaces/*ID*/ipFilter/filterAddresses		Type	Resource
Function	Access IP filtering list			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<IPFilterAddressList>	
PUT		<IPFilterAddressList>	<ResponseStatus>	
POST		<IPFilterAddress>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	The IP filter address list allows addresses to be added and removed from the list, or the entire list to be uploaded at once.			

IPFilterAddressList XML Block

```
<IPFilterAddressList version="1.0" xmlns="urn:psialliance-org">
      <IPFilterAddress/>           <!-- opt -->
</IPFilterAddressList>
```

9.5.8 /PSIA/System/Network/interfaces/<ID>/ipFilter/filterAddresses/<ID>

URI	/PSIA/System/Network/interfaces/*ID*/ipFilter/filterAddresses/*ID*		Type	Resource
Function	Access a particular IP filtering entry.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<IPFilter>	
PUT		<IPFilter>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	If the <permissionType> tag is not provided as a direct child of <IPFilter>, the <permissionType> tag must be provided for each <IPFilterAddress>. Since the ordering of the filters can change the behavior, filtering will be applied consecutively starting with the first <IPFilterAddress> in the list. The <bitMask> field is applied to the corresponding IP address to identify a range of addresses. It indicates the number of '1' bits used to mask the address. For example: '24' would correspond to a subnet mask of 255.255.255.0 and '32' would correspond to a subnet mask of 255.255.255.255 (a single IP address) for IPv4. If <addressFilterType> refers to "mask", the <AddressMask> block must be provided in place of the <AddressRange> block. If it refers to "range", the <Range> block must be provided in place of the <AddressMask> block. Use of IPv4 or IPv6 addresses depends on the value of the <ipVersion> field in /PSIA/System/Network/interfaces/*ID*/ipAddress.			

IPFilterAddress XML Block

```
<IPFilterAddress version="1.0" xmlns="urn:psialliance-org">
      <id>                                        <!-- req, xs:string;id -->
            </id>
      <permissionType>                    <!-- dep, xs:string, "deny,allow" -->
      </permissionType>
      <addressFilterType>           <!-- req, xs:string, "mask,range" -->
      </addressFilterType>
      <AddressRange>                        <!-- dep, depends on <addressFilterType> -->
            <startIPAddress>              <!-- dep, xs:string -->
      </startIPAddress>
            <endIPAddress>                <!-- dep, xs:string -->
      </endIPAddress>
            <startIPv6Address>            <!-- dep, xs:string -->
      </startIPv6Address>
            <endIPv6Address>              <!-- dep, xs:string -->
      </endIPv6Address>
      </AddressRange>
      <AddressMask>                         <!-- dep, depends on <addressFilterType> -->
            <ipAddress>                           <!-- dep, xs:string -->
            </ipAddress>
```

```
            <ipv6Address>                <!-- dep, xs:string -->
      </ipv6Address>
            <bitMask>                          <!-- req, xs:string -->
            </bitMask>
      </AddressMask>
</IPFilterAddress>
```

9.5.9 /PSIA/System/Network/interfaces/<ID>/snmp

URI	/PSIA/System/Network/interfaces/*ID*/snmp		Type	Resource
Function	SNMP settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<SNMP>	
PUT		<SNMP>	<ResponseStatus>	
Notes	At least one of the <SNMPv2c> block or <SNMPAdvanced> block must be provided.			

SNMP XML Block

```
<SNMP version="1.0" xmlns="urn:psialliance-org">
      <SNMPv2c/>                  <!-- dep, either <SNMPv2c> or <SNMPAdvanced> is required -->
      <SNMPAdvanced/>       <!-- dep -->
</SNMP>
```

9.5.10 /PSIA/System/Network/interfaces/<ID>/snmp/v2c

URI	/PSIA/System/Network/interfaces/*ID*/snmp/v2c		Type	Resource
Function	SNMP V2C parameters.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<SNMPv2c>	
PUT		<SNMPv2c>	<ResponseStatus>	
Notes	SNMP v2c configuration includes SNMP notification parameters and a set of SNMP trap receivers. SNMP v2c comprises SNMP v2 without the controversial new SNMP v2 security model, using instead the simple community-based security scheme of SNMP v1			

SNMPv2c XML Block

```
<SNMPv2c version="1.0" xmlns="urn:psialliance-org">
      <notificationEnabled>         <!-- req, xs:boolean -->      </notificationEnabled>
      <SNMPTrapReceiverList/>       <!-- opt -->
</SNMPv2c>
```

9.5.11 /PSIA/System/Network/interfaces/<ID>/snmp/v2c/trapReceivers

URI	/PSIA/System/Network/interfaces/*ID*/snmp/v2c/trapReceivers		Type	Resource
Function	SNMP trap receivers list.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<SNMPTrapReceiverList>	
PUT		<SNMPTrapReceiverList>	<ResponseStatus>	
POST		<SNMPTrapReceiver>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	It is possible to PUT the entire list at once.			

SNMPTrapReceiverList XML Block

```
<SNMPTrapReceiverList version="1.0" xmlns="urn:psialliance-org">
      <SNMPTrapReceiver/>    <!-- opt -->
</SNMPTrapReceiverList>
```

9.5.12 /PSIA/System/Network/interfaces/<ID>/snmp/v2c/trapReceivers/<ID>

URI	/PSIA/System/Network/interfaces/*ID*/snmp/v2c/trapReceivers/*ID*		Type	Resource
Function	SNMP trap receiver information.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<SNMPTrapReceiver>	
PUT		<SNMPTrapReceiver>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	<communityString> format must conform to the SNMPv2c standard.			

SNMPTrapReceiver XML Block

```
<SNMPTrapReceiver version="1.0" xmlns="urn:psialliance-org">
      <id>                                <!-- req, xs:string;id -->
      </id>
      <ReceiverAddress/>            <!-- req -->
      <notificationType>            <!-- req, xs:string, "trap,inform" -->
      </notificationType>
```

```
      <communityString>            <!-- opt, xs:string -->
      </communityString>
</SNMPTrapReceiver>
```

9.5.13 /PSIA/System/Network/interfaces/<ID>/snmp/advanced

URI	/PSIA/System/Network/interfaces/*ID*/snmp/advanced		Type	Resource
Function	Advanced SNMP settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<SNMPAdvanced>	
PUT		<SNMPAdvanced>	<ResponseStatus>	
Notes	<localEngineID> is a hexadecimal string indicating the local device engine ID. <authenticationNotificationEnabled> indicates if SNMP authentication failure notification is enabled on the device. <SNMPNotificationFilterList> is a list to filter traps based on OIDs.			

SNMPAdvanced XML Block

```
<SNMPAdvanced version="1.0" xmlns="urn:psialliance-org">
      <localEngineID>       <!-- req, xs:hexBinary, see RFC2571 -->
      </localEngineID>
      <authenticationNotificationEnabled>
            <!-- opt, xs:boolean -->
      </authenticationNotificationEnabled>
      <SNMPUserList/>                                          <!-- opt -->
      <SNMPNotificationFilterList/>              <!-- opt -->
      <notificationEnabled>                            <!-- opt, xs:boolean -->
      </notificationEnabled>
      <SNMPNotificationReceiverList/>            <!-- opt -->
</SNMPAdvanced>
```

9.5.14 /PSIA/System/Network/interfaces/<ID>/snmp/advanced /users

URI	/PSIA/System/Network/interfaces/*ID*/snmp/advanced/users		Type	Resource
Function	SNMP users.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<SNMPUserList>	
PUT		<SNMPUserList>	<ResponseStatus>	
POST		<SNMPUser>	<ResponseStatus>	
DELETE			<ResponseStatus>	

Notes	Defines the set of SNMP users and their permissions.

SNMPUserList XML Block

```
<SNMPUserList version="1.0" xmlns="urn:psialliance-org">
      <SNMPUser/>           <!-- opt -->
</SNMPUserList>
```

9.5.15 /PSIA/System/Network/interfaces/<ID>/snmp/advanced/users/<ID>

URI	/PSIA/System/Network/interfaces/*ID*/snmp/advanced/users/*ID*		Type	Resource
Function	SNMP user settings.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<SNMPUser>	
PUT		<SNMPUser>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	<remoteEngineID> indicates the remote SNMP entity to which the user is connected. <snmpAuthenticationMethod> indicates the authentication method used. <snmpAuthenticationKey> defines the authentication key if encryption is used for <snmpAuthenticationMethod>. <snmpAuthenticationPassword> optional password used to calculate the <snmpAuthenticationKey> value if encryption is used for <snmpAuthenticationMethod>. <snmpPrivacyMethod> indicates if messages are protected from disclosure, and if so, the type of privacy protocol used. <snmpPrivacyKey> defines the privacy key if encryption is used for <snmpPrivacyMethod>. <snmpPrivacyPassword> optional password used to calculate the <snmpPrivacyKey> value if encryptions is used for <snmpPrivacyMethod>.			

SNMPUser XML Block

```
<SNMPUser version="1.0" xmlns="urn:psialliance-org">
      <id>                        <!-- req, xs:string;id -->          </id>
      <userName>                  <!-- req, xs:string -->             </userName>
      <remoteEngineID>      <!-- req, xs:hexBinary -->          </remoteEngineID>
      <snmpAuthenticationMethod>
            <!-- req, xs:string, "MD5,SHA,none" -->
      </snmpAuthenticationMethod>
      <snmpAuthenticationKey>       <!-- dep, xs:string -->
      </snmpAuthenticationKey>
      <snmpAuthenticationPassword>
            <!-- dep, xs:sring, see RFC3414 -->
```

```
      </snmpAuthenticationPassword>
      <snmpPrivacyMethod>          <!-- req, xs:string, "DES,none" -->   </snmpPrivacyMethod>
      <snmpPrivacyKey>                    <!-- dep, xs:string -->
      </snmpPrivacyKey>
      <snmpPrivacyPassword>        <!-- dep, xs:string, see RFC3414 --> </snmpPrivacyPassword>
</SNMPUser>
```

9.5.16 /PSIA/System/Network/interfaces/<ID>/snmp/advanced/ notificationFilters

URI	/PSIA/System/Network/interfaces/*ID*/snmp/advanced/ notificationFilters		Type	Resource
Function	SNMP notification filters.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<SNMPNotificationFilterList>	
PUT		<SNMPNotificationFilterList>	<ResponseStatus>	
POST		<SNMPNotificationFilter>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	Manages a list of notification filters for SNMP v2 or v3.			

SNMPNotificationFilterList XML Block

```
<SNMPNotificationFilterList version="1.0" xmlns="urn:psialliance-org">
      <SNMPNotificationFilter/>            <!-- req -->
</SNMPNotificationFilterList>
```

9.5.17 /PSIA/System/Network/interfaces/<ID>/snmp/advanced/ notificationFilters/<ID>

URI	/PSIA/System/Network/interfaces/*ID*/snmp/advanced/ notificationFilters/*ID*		Type	Resource
Function	SNMP notification filter settings.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<SNMPNotificationFilter>	
PUT		<SNMPNotificationFilter>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	<oidSubtree> specifies the OID for which notifications are sent or blocked. <filterAction> indicates whether notifications regarding the OID are sent to the trap recipients.			

SNMPNotificationFilter XML Block

```
<SNMPNotificationFilter version="1.0" xmlns="urn:psialliance-org">
      <id>                  <!-- req, xs:string;id -->           </id>
      <filterName>   <!-- req, xs:string -->              </filterName>
      <OIDSubtreeList>                    <!-- opt -->
            <OID>
                  <oidSubtree>          <!-- req, xs:string -->
                  </oidSubtree>
                  <filterAction>        <!-- req, xs:string, "included,excluded" -->
      </filterAction>
            </OID>
      </OIDSubtreeList>
</SNMPNotificationFilter>
```

9.5.18 /PSIA/System/Network/interfaces/<ID>/snmp/advanced/ notificationReceivers

URI	/PSIA/System/Network/interfaces/*ID*/snmp/advanced/ notificationReceivers		Type	Resource
Function	SNMP notification receivers.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<SNMPNotificationReceiverList>	
PUT		<SNMPNotificationReceiverList>	<ResponseStatus>	
POST		<SNMPNotificationReceiver>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	Manage the list of SNMP notification receivers for v2 or v3.			

SNMPNotificationReceiverList XML Block

```
<SNMPNotificationReceiverList version="1.0" xmlns="urn:psialliance-org">
      <SNMPNotificationReceiver>          <!-- opt -->
</SNMPNotificationReceiverList>
```

9.5.19 /PSIA/System/Network/interfaces/<ID>/snmp/advanced/ notificationReceivers/<ID>

URI	/PSIA/System/Network/interfaces/*ID*/snmp/advanced/ notificationReceivers/*ID*		Type	Resource
Function	SNMP notification receiver settings.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<SNMPNotificationReceiver>	
PUT		<SNMPNotificationReceiver>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	<notificationType> indicates whether this receiver entry is for a trap or an inform. <userID> must correspond to a user ID in /PSIA/System/Network/interfaces/*ID*/snmp/advanced/users/*ID*. <securityType> defines the security level attached to the user. The "authentication" option will authenticate SNMP messages and ensure the origin is authenticated. The "privacy" option authenticates and encrypts the SNMP messages. <filterName> associates a filter if <filterEnabled> is true. <timeout> indicates the amount of time (seconds) the device waits before re-sending informs. <retries> indicates the number of times the device re-sends an inform request.			

SNMPNotificationReceiver XML Block

```
<SNMPNotificationReceiver version="1.0" xmlns="urn:psialliance-org">
      <ReceiverAddress/>            <!-- req -->
      <notificationType>            <!-- req, xs:string, "trap,inform" -->
      </notificationType>
      <userID>                              <!-- req, xs:string -->
            </userID>
      <securityType>
            <!-- req, xs:string, "noauthentication,authentication,privacy" -->
      </securityType>
      <filterEnabled>       <!-- req, xs:boolean -->
      </filterEnabled>
      <filterName>          <!-- req, xs:integer -->
      </filterName>
      <timeout>                     <!-- opt, xs:integer, seconds -->
      </timeout>
      <retries>                     <!-- opt, xs:integer -->
      </retries>
</SNMPNotificationReceiver>
```

9.5.20 /PSIA/System/Network/interfaces/<ID>/snmp/v3

URI	/PSIA/System/Network/interfaces/*ID*/snmp/v3		Type	Resource
Function	SNMP v3 settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<SNMPAdvanced>	
PUT		<SNMPAdvanced>	<ResponseStatus>	
Notes	This resource is an alias to /PSIA/System/Network/interfaces/*ID*/snmp/advanced. The <snmpAuthenticationPassword> and <snmpPrivacyPassword> tags are optionally used if the device implementation chooses to calculate the corresponding keys based on a password (as in RFC3414). In this case, the <snmpAuthenticationKey> and <snmpPrivacyKey> may or may not be provided. The <localEngineID> tag is used for "trap" messages and the <remoteEngineID> tag is used for "inform" messages.			

9.5.21 /PSIA/System/Network/interfaces/<ID>/qos

URI	/PSIA/System/Network/interfaces/*ID*/qos		Type	Resource
Function	This function is used to set the QoS setting for the device.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<QoS>	
PUT		<QoS>	<ResponseStatus>	
Notes	At least one of <CoSList> or <DSCPList> must be provided.			

QoS XML Block

```
<QoS version="1.0" xmlns="urn:psialliance-org">
      <CoSList/>            <!-- dep -->
      <DSCPList/>           <!-- dep -->
</QoS>
```

9.5.22 /PSIA/System/Network/interfaces/<ID>/qos/cos

URI	/PSIA/System/Network/interfaces/*ID*/qos/cos		Type	Resource
Function	Class of Service (CoS) settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<CoSList>	
PUT		<CoSList>	<ResponseStatus>	
POST		<CoS>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	A list of class of service parameter blocks is specified for each type of traffic: device management, command and control, video and audio streaming. Devices may extend the set of traffic types.			

CoSList XML Block

```
<CoSList version="1.0" xmlns="urn:psialliance-org">
      <CoS/>         <!-- opt -->
</CoSList>
```

9.5.23 /PSIA/System/Network/interfaces/<ID>/qos/cos/<ID>

URI	/PSIA/System/Network/interfaces/*ID*/qos/cos/*ID*		Type	Resource
Function	Class of service settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<CoS>	
PUT		<CoS>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	<trafficType> determines which kind of traffic the settings apply to.			

CoS XML Block

```
<CoS version="1.0" xmlns="urn:psialliance-org">
      <id>                  <!-- req, xs:string;id -->           </id>
      <enabled>             <!-- req, xs:boolean -->             </enabled>
      <priority>            <!-- req, xs:integer -->             </priority>
      <vlanID>              <!-- req, xs:string -->              </vlanID>
      <trafficType>
            <!-- req, xs:string, "devicemanagement,commandcontrol,video,audio" -->
      </trafficType>
</CoS>
```

9.5.24 /PSIA/System/Network/interfaces/<ID>/qos/dscp

URI	/PSIA/System/Network/interfaces/*ID*/qos/dscp		Type	Resource
Function	Differentiated Services (DiffServ) settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<DSCPList>	
PUT		<DSCPList>	<ResponseStatus>	
POST		<DSCP>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	A list of DSCP parameter blocks is specified for each type of traffic: device management, command and control, video and audio streaming. Devices may extend the set of traffic types.			

DSCPList XML Block

```
<DSCPList version="1.0" xmlns="urn:psialliance-org">
      <DSCP/>        <!-- opt -->
</DSCPList>
```

9.5.25 /PSIA/System/Network/interfaces/<ID>/qos/dscp/<ID>

URI	/PSIA/System/Network/interfaces/*ID*/qos/dscp/*ID*		Type	Resource
Function	DSCP entry settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<DSCP>	
PUT		<DSCP>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	<trafficType> determines which kind of traffic the settings apply to.			

DSCP XML Block

```
<DSCP version="1.0" xmlns="urn:psialliance-org">
      <id>                          <!-- req, xs:string;id -->           </id>
      <enabled>                     <!-- req, xs:boolean -->             </enabled>
      <priorityValue>       <!-- req, xs:integer, 6 bits - refer to RFC2474 -->
      </priorityValue>
      <trafficType>
            <!-- req, xs:string, "devicemanagement,commandcontrol,video,audio" -->
      </trafficType>
</DSCP>
```

9.5.26 /PSIA/System/Network/interfaces/<ID>/discovery

URI	/PSIA/System/Network/interfaces/*ID*/discovery		Type	Resource
Function	Device discovery settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<Discovery>	
PUT		<Discovery>	<ResponseStatus>	
Notes	Use of IPv4 or IPv6 addresses depends on the value of the <ipVersion> field in /PSIA/System/Network/interfaces/*ID*/ipAddress. <portNo> is the port number for the multicast discovery address. <ttl> is the time to live for multicast discovery packets.			

Discovery XML Block

```
<Discovery version="1.0" xmlns="urn:psialliance-org">
      <UPnP>                        <!-- opt -->
            <enabled>               <!-- req, xs:boolean -->      </enabled>
      </UPnP>
      <Zeroconf>                    <!-- opt -->
            <enabled>               <!-- req, xs:boolean -->      </enabled>
      </Zeroconf>
      <MulticastDiscovery>  <!-- opt -->
            <enabled>               <!-- req, xs:boolean -->      </enabled>
            <ipAddress>             <!-- dep, xs:string -->       </ipAddress>
            <ipv6Address>  <!-- dep, xs:string -->       </ipv6Address>
            <portNo>                <!-- req, xs:integer -->      </portNo>
            <ttl>                   <!-- req, xs:integer -->      </ttl>
      </MulticastDiscovery>
</Discovery>
```

9.5.27 /PSIA/System/Network/interfaces/<ID>/syslog

URI	/PSIA/System/Network/interfaces/*ID*/syslog		Type	Resource
Function	Syslog settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<Syslog>	
PUT		<Syslog>	<ResponseStatus>	
Notes	Configure the system settings.			

Syslog XML Block

```
<Syslog version="1.0" xmlns="urn:psialliance-org">
      <enabled>                         <!-- req, xs:boolean -->      </enabled>
      <SyslogServerList/>   <!-- opt -->
</Syslog>
```

9.5.28 /PSIA/System/Network/interfaces/<ID>/syslog/servers

URI	/PSIA/System/Network/interfaces/*ID*/syslog/servers		Type	Resource
Function	Syslog server list.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<SyslogServerList>	
PUT		<SyslogServerList>	<ResponseStatus>	
POST		<SyslogServer>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	Manage a set of syslog servers that receive logging notifications.			

SyslogServerList XML Block

```
<SyslogServerList version="1.0" xmlns="urn:psialliance-org">
      <SyslogServer/>       <!-- opt -->
</SyslogServerList>
```

9.5.29 /PSIA/System/Network/interfaces/<ID>/syslog/servers/<ID>

URI	/PSIA/System/Network/interfaces/*ID*/syslog/servers/*ID*		Type	Resource
Function	Syslog server settings.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<SyslogSever>	
PUT		<SyslogServer>	<ResponseStatus>	
DELETE			<ResponseStatus>	
Notes	Depending on the value of <addressingFormatType>, either the <hostName> or the IP address fields will be used to locate the NTP server. Use of IPv4 or IPv6 addresses depends on the value of the <ipVersion> field in /PSIA/System/Network/interfaces/*ID*/ipAddress. <facilityType> indicates the facility to store syslog messages. See RFC3164. <severity> indicates the minimum log severity for which to send a syslog message. See RFC3164.			

SyslogServer XML Block

```
<SyslogServer version="1.0" xmlns="urn:psialliance-org">
      <id>                  <!-- req, xs:string;id -->          </id>
      <addressingFormatType>
            <!-- req, xs:string, "ipaddress,hostname" -->
      </addressingFormatType>
      <hostName>            <!-- dep, xs:string -->             </hostName>
      <ipAddress>           <!-- dep, xs:string -->             </ipAddress>
```

```
      <ipv6Address>  <!-- dep, xs:string -->              </ipv6Address>
      <portNo>              <!-- opt, xs:integer -->             </portNo>
      <facilityType> <!-- req, xs:string, see RFC3164 -->        </facilityType>
      <severity>            <!-- opt, xs:string, see RFC3164 -->        </severity>
</SyslogServer>
```

9.5.30 Examples

Example: Getting the Network Settings

```
GET /PSIA/System/Network HTTP/1.1
…
HTTP/1.1 200 OK
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<NetworkInterfaceList version="1.0" xmlns="urn:psialliance-org">
      <NetworkInterface>
            <id>1</id>
            <IPAddress>
                  <ipVersion>v4</ipVersion>
                  <addressingType>static</addressingType>
                  <ipAddress>3.137.217.220</ipAddress>
                  <subnetMask>255.255.255.0</subnetMask>
                  <DefaultGateway>
                        <ipAddress>3.137.217.0</ipAddress>
                  </DefaultGateway>
                  <PrimaryDNS>
                        <ipAddress>3.137.218.37</ipAddress>
                  </PrimaryDNS>
                  <SecondaryDNS>
                        <ipAddress>3.137.217.15</ipAddress>
                  </SecondaryDNS>
            </IPAddress>
      </NetworkInterface>
      <NetworkInterface>
            <id>2</id>
            <IPAddress>
                  <ipVersion>v4</ipVersion>
                  <addressingType>dynamic</addressingType>
            </IPAddress>
            <Wireless>
                  <enabled>true</enabled>
                  <wirelessNetworkMode>intrastructure</wirelessNetworkMode>
                  <WirelessSecurity>
                        <securityMode>WPA-personal</securityMode>
                        <WPA>
                              <algorithmType>AES</algorithmType>
```

```
                                    <sharedKey>ac34587bc8a8fff7a</sharedKey>
                            </WPA>
                        </WirelessSecurity>
                </Wireless>
        </NetworkInterface>
</NetworkInterfaceList>
```

Example: Setting the IP Address

```
PUT /PSIA/System/Network/interfaces/1/ipAddress HTTP/1.1
…
HTTP/1.1 200 OK
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<IPAddress version="1.0" xmlns="urn:psialliance-org">
        <ipVersion> v4</ipVersion>
        <addressingType>static </addressingType>
        <ipAddress>3.137.217.220</ipAddress>
        <subnetMask>255.255.255.0</subnetMask>
        <DefaultGateway>
                <ipAddress>3.137.217.0</ipAddress>
        </DefaultGateway>
        <PrimaryDNS>
                <ipAddress>3.137.218.37</ipAddress>
        </PrimaryDNS>
        <SecondaryDNS>
                <ipAddress>3.137.217.15</ipAddress>
        </SecondaryDNS>
</IPAddress>
```

9.6 /PSIA/System/IO

URI	/PSIA/System/IO		Type	Service
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<IOPortList>	
Notes	The allocation of IDs between input and output ports must be unique.			

IOPortList XML Block

```
<IOPortList version="1.0" xmlns="urn:psialliance-org">
      <IOInputPortList/>            <!-- opt -->
      <IOOutputPortList/>    <!-- opt -->
</IOPortList>
```

9.6.7 /PSIA/System/IO/status

URI	/PSIA/System/IO/status		Type	Resource
Function	Query the IO status.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<IOPortStatusList>	
Notes	<ioPortID> refers to /PSIA/System/IO/inputs/*ID* or /PSIA/System/IO/outputs/*ID*. The port IDs are guaranteed to be unique across input and output ports. <ioState> indicates whether the input port is active or inactive. In most applications, a high signal is considered active.			

IOPortStatus XML Block

```
<IOPortStatusList version="1.0" xmlns="urn:psialliance-org">
      <IOPortStatus>         <!-- opt -->
            <ioPortID>             <!-- req, xs:string;id -->
            </ioPortID>
            <ioPortType>   <!-- req, xs:string, "input,output" -->
      </ioPortType>
            <ioState>              <!-- req, xs:string, "active,inactive" -->
      </ioState>
      </IOPortStatus>
</IOPortStatusList>
```

9.6.8 /PSIA/System/IO/inputs

URI	/PSIA/System/IO/inputs		Type	Resource
Function	Access input ports.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	

GET			<IOInputPortList>
Notes	IO inputs are hardwired, meaning that the inputs are statically allocated by the device and cannot be created or deleted.		

IOInputPortList XML Block

```
<IOInputPortList version="1.0" xmlns="urn:psialliance-org">
      <IOInputPort/>        <!-- opt -->
</IOInputPort>
```

9.6.9 /PSIA/System/IO/inputs/<ID>

URI	/PSIA/System/IO/inputs/*ID*		Type	Resource
Function	Access a particular input port.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<IOInputPort>	
PUT		<IOInputPort>	<ResponseStatus>	
Notes	<triggeringType> indicates the signal conditions to trigger the input port. Rising/Falling refer to a rising/falling edge of a signal. High/Low will continuously trigger for the duration of the high/low input signal.			

IOInputPort XML Block

```
<IOInputPort version="1.0"    xmlns="urn:psialliance-org">
      <id>                          <!-- req, xs:string;id -->
                  </id>
      <triggeringType>      <!-- req, xs:string, "high,low,rising,falling" -->
      </triggeringType>
</IOInputPort>
```

9.6.10 /PSIA/System/IO/inputs/<ID>/status

URI	/PSIA/System/IO/inputs/*ID*/status		Type	Resource
Function	Query the status of an input port.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<IOPortStatus>	
Notes	See /PSIA/System/IO/status for an explanation of the fields.			

9.6.11 /PSIA/System/IO/outputs

URI	/PSIA/System/IO/outputs		Type	Resource
Function	Access output ports.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	

GET			<IOOutputPortList>
Notes	IO outputs are hardwired, meaning that the inputs are statically allocated by the device and cannot be created or deleted.		

IOOutputPortList XML Block

```
<IOOutputPortList version="1.0" xmlns="urn:psialliance-org">
      <IOOutputPort/>       <!-- opt -->
</IOOutputPort>
```

9.6.12 /PSIA/System/IO/outputs/<ID>

URI	/PSIA/System/IO/outputs/*ID*		Type	Resource
Function	Access a particular output port.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<IOOutputPort>	
PUT		<IOOutputPort>	<ResponseStatus>	
Notes	<PowerOnState> defines the output port configuration when the device is powered on. <defaultState> is the default output port signal when it is not being triggered. <outputState> is the output port signal when it is being triggered. Pulse will cause the output port to send a signal (opposite of the <defaultState>) for a duration specified by the <pulseDuration> tag. <pulseDuration> is the duration of a pulse output port signal when it is being triggered. It must be provided if the <outputState> is "pulse". <actionMapping> is used in interfaces that allow configuration of "On" and "Off" for "High" and "Low" signals.			

IOOutputPort XML Block

```
<IOOutputPort version="1.0" xmlns="urn:psialliance-org">
      <id>                                  <!-- req, xs:string;id -->
      </id>
      <PowerOnState>                <!-- req -->
            <defaultState>          <!-- req, xs:string, "high,low" -->
      </defaultState>
            <outputState>           <!-- req, xs:string, "high,low,pulse" -->    </outputState>
            <pulseDuration>         <!-- dep, xs:integer, milliseconds -->
      </pulseDuration>
      </PowerOnState>
      <ManualControl>               <!-- opt -->
            <actionMapping>
                  <!-- req, xs:string, "high,low": ON maps to high / ON maps to low -->
            </actionMapping>
      </ManualControl>
</IOOutputPort>
```

9.6.13 /PSIA/System/IO/outputs/<ID>/trigger

URI	/PSIA/System/IO/outputs/*ID*/trigger		Type	Resource
Function	Manually trigger an output port.			
Methods	Query String(s)	Inbound Data	Return Result	
PUT	outputState pulseDuration	<IOPortData>	<ResponseStatus>	
Notes	Either the inbound data or query string values are used. The IO output port is toggled to a high or low signal accordingly. If the <outputState> refers to pulse, then the <pulseDuration> tag must be provided and the output port will be triggered to the specified state for the duration specified by <pulseDuration>.			

IOPortData XML Block

```
<IOPortData xmlns="urn:psialliance-org">
      <outputState>         <!-- req, xs:string, "high,low,pulse" -->    </outputState>
      <pulseDuration>       <!-- dep, xs:integer, milliseconds -->
      </pulseDuration>
</IOPortData>
```

9.6.14 /PSIA/System/IO/outputs/<ID>/status

URI	/PSIA/System/IO/inputs/*ID*/status		Type	Resource
Function	Query the status of an output port.			
Methods	Query String(s)	Inbound Data	Return Result	
GET			<IOPortStatus>	
Notes	See /PSIA/System/IO/status for an explanation of the fields.			

9.6.15 IO Port Examples

Example: Set up IO Port Triggering

NOTE: The following example requires that input port event detection and output port triggering be enabled and scheduled with /PSIA/Custom/Event/triggers and /PSIA/Custom/Event/schedule beforehand.

The following commands set up one device input port and two device output ports (the number of IO ports is device-dependent) in the following manner:

- Input port 111 will continuously trigger an event when the input signal is high. The input port should stop triggering this event when the input signal reverts back to low.
- Output port 222 will have a default low signal when not being triggered. When triggered, it will switch to a high signal. The port should automatically revert to a low signal when triggering stops, but in the case that a device cannot support this feature the port can be manually reset
- Output port 333 will have a default low signal when not being triggered. When triggered, it will send a "pulse" of the opposite signal - high, in this case - for a duration of 3 seconds and then switch back to a low signal.

```
PUT /PSIA/System/IO/inputs/111 HTTP/1.1
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<IOInputPort>
       <triggeringType>high</triggeringType>
</IOInputPort>
```

```
PUT /PSIA/System/IO/outputs/222 HTTP/1.1
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<IOOutputPort>
       <PowerOnState>
              <defaultStateType>low</defaultStateType>
              <outputStateType>high</outputStateType>
   </PowerOnState>
</IOOutputPort>
```

```
PUT /PSIA/System/IO/outputs/333 HTTP/1.1
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<IOOutputPort>
       <PowerOnState>
              <defaultStateType>low</defaultStateType>
              <outputStateType>pulse</outputStateType>
              <pulseDuration>3000</pulseDuration>
   </PowerOnState>
</IOOutputPort>
```

Example: Manually Trigger and Reset an Output Port

Use the following command to manually set to a low signal. Note that this feature has no effect on future event detection and triggering – e.g. if output port 1 is automatically triggered in the future, it will override the behavior set here.

```
PUT /PSIA/System/IO/outputs/222/trigger HTTP/1.1
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<IOPortData xmlns="urn:psialliance-org">
```

```
      <outputState>low</outputState>
</IOPortData>
```

or, the same without the XML payload:

```
PUT /PSIA/System/IO/outputs/222/trigger?outputState=low HTTP/1.1
```

9.7 /PSIA/System/Audio

This Service, and its accompanying resources, are defined in the PSIA IP Media Device specification.

9.8 /PSIA/System/Video

This Service, and its accompanying resources, are defined in the PSIA IP Media Device specification.

9.9 /PSIA/System/Serial

URI	/PSIA/System/Serial		Type	Service
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
Notes	Serial line service.			

9.9.7 /PSIA/System/Serial/ports

URI	/PSIA/System/Serial/ports		Type	Resource
Function	List of serial ports supported by the device.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<SerialPortList>	
Notes	Since serial ports are resources that are defined by the hardware configuration of the device, they cannot be created or deleted.			

SerialPortList XML Block

```
<SerialPortList version="1.0" xmlns="urn:psialliance-org">
      <SerialPort/>  <!-- opt -->
</SerialPortList>
```

9.9.8 /PSIA/System/Serial/ports/<ID>

URI	/PSIA/System/Serial/ports/*ID*		Type	Resource
Function	Serial port			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET			<SerialPort>	
PUT		<SerialPort>	<ResponseStatus>	
Notes	Access to the serial port parameters. <serialPortType> set the type of port; RS232, RS485, etc. <direction> indicates whether the port is bidirectional. <duplexMode> indicates whether the serial port runs in full or half duplex mode.			

SerialPort XML Block

```
<SerialPort version="1.0" xmlns="urn:psialliance-org">
      <id>                          <!-- req, xs:string;id -->
                  </id>
      <enabled>                     <!-- req, xs:boolean -->
                  </enabled>
      <serialPortType>      <!-- req, xs:string, "RS485,RS422,RS232" -->
      </serialPortType>
      <duplexMode>          <!-- req, xs:string, "half,full" -->
      </duplexMode>
```

```
      <direction>                  <!-- req, xs:string, "monodirectional,bdirectional" -->
      </direction>
      <baudRate>                   <!-- req, xs:integer -->
                  </baudRate>
      <dataBits>                   <!-- req, xs:integer -->
                  </dataBits>
      <parityType>          <!-- req, xs:string, "none,even,odd,mark,space" -->  </parityType>
      <stopBits>                   <!-- req, xs:string, "1,1.5,2" -->
      </stopBits>
</SerialPort>
```

9.9.9 /PSIA/System/Serial/ports/<ID>/command

URI	/PSIA/System/Serial/ports/*ID*/command		Type	Resource
Function	Send a command to a serial port.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
PUT	chainNo	<SerialCommand> Raw Data	<ResponseStatus>	
Notes	If the IP device is an analog-to-digital encoder and is connected to analog PTZ-enabled camera(s), it is the device's responsibility to relay the request to the appropriate serial interface based on the <chainNo> tag or query string. If the IP device is itself a PTZ-enabled digital camera, it is the device's responsibility to address the correct serial interface for the corresponding PTZ command. The serial command can either be encapsulated in the <command> field, in which case the data should be encoded in hexadecimal notation, or the data can be uploaded directly as the HTTP payload, in which case the content type should be `application/octet-stream`. In this case, the chainNo query string parameter should be used.			

SerialCommand XML Block

```
<SerialCommand version="1.0" xmlns="urn:psialliance-org">
      <chainNo>             <!-- opt, xs:string -->       </chainNo>
      <command>             <!-- req, xs:hexBinary -->    </command>
</SerialCommand>
```

Example

Send the command using an XML block:

```
PUT /PSIA/System/Serial/ports/999/command HTTP/1.1
Content-Type: application/xml; charset="UTF-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8"?>
<SerialCommand>
      <chainNo>0</chainNo>
      <command>ab45be8778cd</command>
```

```
</SerialCommand>
```

Send the command using query strings and a binary payload:

```
PUT /PSIA/System/Serial/ports/999/command?chainNo=1 HTTP/1.1
Content-Type: application/octet-stream
Content-Length: xxx

(…Raw bytes of command follow here…)
```

9.10.1 /PSIA/System/Battery

URI	/PSIA/System/Battery		Type	Resource
Function	Interface for retrieving Battery operating attributes			
Methods	Query String(s)	Inbound Data	Return Result	
GET	No	None.	<BatteryInfo> Raw schema document.	
Notes	This resource identifies the attributes of battery components within a device or system. The resource document identifies the number, composition, status and notifying thresholds of the resident battery components. NOTE: Since battery assemblies are intrinsic, the PUT, POST and DELETE functions are ***not*** allowed and any of these operations will result in a "405 Method Not Allowed' HTTP status response.			

The '/PSIA/System/Battery' resource identifies the battery properties/attribuites for battery components on/in a device or system. These attributes are both static and operational in nature. The charge threshold levels, critical and low, may be 'set' via PUT's to the '/PSIA/System/Battery/<ID>' resource (see next section). However, all the attributes in this resource are immutable. Since battery attributes are innate/intrinsic to a node, PUT, POST and DELETE operations are disallowed and will result in an HTTP response with a '405 Method Not Allowed' status code.
The XSD definition for attribute information reported by the '/PSIA/System/Battery' resource is listed below.

```
<?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="BatteryInfo">
        <xs:complexType>
                <xs:attribute name="version" type="xs:string" use="required" />
                <xs:sequence>
                        <xs:element name="NumberOfBatteries" minOccurs="1" maxOccurs="1"
type="xs:integer"/>
                        <xs:element name="Batteries" minOccurs="1" maxOccurs="1"
type="BatteryInfoList"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

```
<xs:complexType name="BatteryInfoList">
        <xs:sequence>
                <xs:element name="BatteryAttributes" minOccurs="1" maxOccurs="unbounded"
type="BatteryProperties"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="BatteryProperties">
        <xs:sequence>
                <xs:element name="BatteryID" minOccurs="1" maxOccurs="1" type="xs:integer"/>
                <xs:element name="BatteryDescription" minOccurs="1" maxOccur="1"
                      type="xs:string"/>
                <xs:element name="BatteryMounting" minOccurs="1" maxOccurs="1"
                      type="BatteryMountType"/>
                <xs:element name="BatteryCharging" minOccurs="1" maxOccurs="1"
                      type="xs:boolean"/>
                <xs:element name="BatteryChargeMetric" minOccurs="1" maxOccurs="1"
                      type="xs:ChargeMetric"/>
                <xs:element name="BatteryChargeLevel" minOccurs="1" maxOccurs="1"
                      type="xs:float"/>
                <xs:element name"BatteryLowThreshold" minOccurs="1" maxOccurs="1"
                      type="xs:float"/>
                <xs:element name"BatteryCriticalThreshold" minOccurs="1" maxOccurs="1"
                      type="xs:float"/>
                <xs:element name="BatteryComponents" minOccurs="1" maxOccurs="1"
                      type="xs:integer"/>
                <xs:element name="BatteryComponentDescription" minOccurs="1" maxOccurs="1"
                      type="xs:string"/>
                <xs:element name="BatteryRole" minOccurs="1" maxOccurs="1" type="BatteryRole"/>
        </xs:sequence>
</xs:complexType>

<xs:simpleType name="BatteryMountType">
        <xs:annotation>
                <xs:documentation xml:lang="en">
                        The following fields define how a battery component
                        is mechanically incorporated into a device or system.
                </xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
                <xs:enumeration value="fixed-enclosure"/>
                <xs:enumeration value="replaceable-internal"/>
                <xs:enumeration value="replaceable-external"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="ChargeMetric">
        <xs:annotation>
                <xs:documentation xml:lang="en">
                        The following string fields define how a battery 'Charge;
                        level is represented: either 'percentage' or (raw) 'voltage'.
                </xs:documentation>
```

```
        </xs:annotation>
        <xs:restriction base="xs:string">
                <xs:enumeration value="percentage"/>
                <xs:enumeration value="voltage"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="BatteryRole">
        <xs:restriction base="xs:string">
                <xs:enumeration value="primary"/>
                <xs:enumeration value="alternate-standby"/>
                <xs:enumeration value="alternate-active"/>
        </xs:restriction>
</xs:simpleType>
```

The above schema definition is compromised of a list of the battery components resident in a device or system. Each battery element in the “Battery” list of “BatteryInfoList” elements is uniquely identified by its “BatteryID” value. This value (i.e. “BatteryID”) MUST be used to modify the “BatteryLowThreshold” and/or “BatteryCriticalTheshold” values when performing a PUT operation to that specific resource (see next section).

9.10.1.1 /PSIA/System/Battery/<ID>

URI	/PSIA/System/Battery/<id>		**Type**	Resource
Function	Interface for setting & retrieving Battery operating attributes fir s specific battery component.			
Methods	**Query String(s)**	**Inbound Data**	**Return Result**	
GET	No	None.	<BatteryOpAttributes> Raw schema document.	
PUT	No	<BatteryOpAttributes schema doc>		
Notes				

```
<?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="BatteryOpAttributes">
      <xs:complexType>
            <xs:attribute name="version" type="xs:string" use="required" />
            <xs:sequence>
                  <xs:element name="BatteryID" minOccurs="1" maxOccurs="1" type="xs:integer"/>
                  <xs:element name"BatteryLowThreshold" minOccurs="1" maxOccurs="1"
                        type="xs:float"/>
                  <xs:element name"BatteryCriticalThreshold" minOccurs="1" maxOccurs="1"
                        type="xs:ifloat"/>
                  <xs:element name="BatteryRole" minOccurs="0" maxOccurs="1" type="BatteryOpRole"/>
            </xs:sequence>
      </xs:complexType>
</xs:element>

<xs:simpleType name="BatteryOpRole">
      <xs:restriction base="xs:string">
            <xs:enumeration value="primary"/>
            <xs:enumeration value="alternate-standby"/>
            <xs:enumeration value="alternate-active"/>
      </xs:restriction>
</xs:simpleType>
</xs:schema>
```

The '/PSIA/System/Battery/<ID>' resource object provides the interface for setting the 'low threshold' and 'critical threshold' values for a particular battery component as identified by the 'ID' value in the resource's URI. The '/PSIA/System/Battery' resource (see previous section), provides the interface for reporting all of the key battery properties for the battery components of a system or device. This interface provides a component-specific interface for modifying the reporting thresholds for a particular battery component. Devices may, or may not, provide the ability to modify the 'BatteryOpRole' attribute. If this element is present during a read operation (i.e. a 'GET'), then the device allows an authorized user to modify the operation role of a battery component. This state, irrespective of being mutable, or not, is reported in the '/PSIA/System/Battery' resource.
Please note that Battery 'state' Events occur when the charge-level traverses one of the above thresholds, and/or a battery's role changes. The battery 'state' event class and type are defined on the Common Metadata and Event Model (CMEM) v2.1 specification, in Appendix 11. Please reference the CMEM v2.1 specification for more details. Additionally, the schema definition for the state information contained in Battery 'state' events is contained, below, in Section 14.1.8 of this specification.

10 Managed Data Transfer (MDT)

PSIA formally specifies two standardized methods for the application-level control of transferring large amounts of data via HTTP. Since data objects may be very large, it is not always practical, or even possible, to transfer the contents of a large data object, in its entirety, in one non-segmented continuous 'stream.' There are two primary issues with transferring very large data objects:

- ***Application Flow Control***: Once a 'GET' is issued by a consumer, HTTP transfers data as fast, and continuously, as TCP allows over a given network. Since the size of many data objects are not known in advance, an application may not have enough buffer space allocated to

adequately, or ideally, receive an entire large data object at the rate it is transferred from the server. A way of managing the rate, and amount, of data transferred is needed for large data objects.

- ***Data Object Subset Access***: Many PSIA REST data objects are comprised of lists of elements. REST in its PSIA-ordained access mechanisms, allows simple retrieval of (i.e. a "GET') an entire data object or an individual list element thereof. However, a practical and efficient method for retrieving multiple elements, of a large list, without multiple 'GETs' is needed to aid overall data management and scalability.

The items listed above are resolved via the methods described below. All PSIA Service Model v2.0 compliant nodes, that serve data, MUST support the first method (Method #1, AFC), and SHOULD support the second method (Method #2, LAM) where it is applicable (i.e. where a data object is comprised of a list element with individual IDs).

10.5 Method #1: Application-Level Flow Control (AFC)

Applications consuming, or sending, an ***entire*** data object of 16KB, or greater in size, MUST use HTTP 'Chunking' (RFC 2616) for segmenting the data transfer into 'chunks.' Each chunk may be less than or equal to 16KB in size. However, an 8KB chunk size is recommended. Using HTTP Chunking, with these chunk size recommendations, provides a better level of transfer control, and overall reliability. The AFC rules are specifically summarized below:

- If the size of any data object being transferred is 16KB, or greater, the sender MUST use HTTP 'chunking' to transfer the entire data object. The 'chunking' protocol is specified in RFC 2616. This requires both the sender and receiver to comply with the segmentation, and header/trailer, rules associated with chunking. The underlying TCP protocol, and socket interface mechanisms, still govern overall session flow at the network layer.
- The chunk size used to transfer the segments comprising the data object may NOT exceed 16KB. A chunk size of 8KB is recommended for each transfer, but for larger data objects, chunk sizes up to 16KB may be used to help reduce session-level overhead. Though it is highly preferred, not all chunk sizes have to be uniform. They are only required to meet the aforementioned size limit.

All PSIA nodes MUST comply with the above rules. This includes senders and receivers irrespective of the web implementation (client versus server).

10.6 Method #2: List Access Management (LAM)

In cases where the data object being retrieved is comprised of a list of elements, each uniquely identified by a 'id' value, of some sort, the list may be retrieved in segments using a standardized notation of providing a simple ID and a count indicator to govern the number of elements being retrieved in a single 'GET'. The format is listed below:

```
<URI>?(startID=<n> OR lastID=<n>)&count=<n>
```

Please note that PSIA currently encourages the use of only 2 standardized value types to be used as 'IDs' in lists:

"localID": This PSIA is provided in 'psiaCommonTypes.xsd' (Section 13.1.6) and each unsigned integer value is used as a local index in the list element.

"globalID": For list items that need global, or system-wide, uniqueness, ITU X.667 compliant 128-bit UUID/GUID values are to be used. This XML type is also provided in 'psiaCommonTypes.xsd.'

For support of legacy implementations, XML ID '*strings'* are allowed but the use of raw strings as ID values has been deprecated. One of the above two ID types *MUST* be used by all Service Model v1.2, and later, PSIA Working Groups.
Using these QSP types, the example URI below accesses a large XML document comprised of port configuration information on a large device:

```
/PSIA/System/USB
```

All port configuration information may be retrieved by using the following URI:

```
GET /PSIA/System/USB
```

Or, a single instance of port's configuration may be retrieved via the following URI (port 4 for example):

```
GET /PSIA/System/USB/4
```

However, if a device had 256 ports, for example, the size of the entire XML document may be prohibitively large. In these cases, the retriever may want to get the entire list, or portions of it, in segments using the 'localID' values to specify the range of elements being accessed. For example, getting the information for ports 16 through 30 would generate the following URI:

```
GET /PSIA/System/USB?startID=16&count=15
```

In the above scenario, the XML schema governing the definition of the information in the REST Resource would be adhered to, except that only the items specified in the list element range would be in each accessed document instance. Please note that the 'range' notation is simple and does not allow 'gaps', or omitted elements within a range. Additionally, only ***one*** range per HTTP request is allowed.
Using this same resource example, a requester may want to get the list in 'segments' of 20 items (or less). The following URIs exemplify using 2 requests to get the first 40 elements in a list:

```
GET /PSIA/System/USB?count=20
. . . .
GET /PSIA/System/USB?lastID=20&count=20
```

I
n the final (above) URI, the requester provides the last element ID received and indicate that is wants the next 20 elements ***after*** that element ID. The use of these QSP types allows requesters to effectively 'walk' lists.
Lists where elements are indexed by UUID/GUIDs would have the following format:

```
GET /PSIA/CSEC/AAA/accessLog?count=10
. . . .
GET /PSIA/CSEC/AAA/accessLog?lastID={1c994ef3-002d-97e6-3f24-
19073cd020b5}&count=10
. . . .
```

Please note that the above REST resource is fictitious and used for example purposes only.

Where legacy systems, or systems constrained by protocol, require the use of raw strings as element IDs, those strings MUST be encapsulated in double-quotes (“) when supplied as a QSP. For example:

```
GET /PSIA/System/volumeTags?count=5
. . .
GET /PSIA/System/volumeTags?lastID=”GDRIVE”&count=5
. . .
GET /PSIA/System/volumeTags?lastID=”1002-9C22E-FMOUNT”&count=5
```

In the above example, the requester is consuming a list 5 elements per request. The elements are identified by raw strings. After the first request, each subsequent request MUST list the last element’s ID value such that the target knows where to resume transmission of data.

10.7 General Format Rules for IDs in Query String Parameters

As mentioned above, certain formatting rules are imposed for the sake of consistency, and clarity, when conveying ID values between nodes. These formatting rules are explicitly listed below for each ID type:

- **“localID”**: These integer values are supplied as ASCII decimal values per the type defined in “***psiaCommonTypes.xsd.***”
- **“globalID”**: These UUID/GUID values are supplied encapsulated in ‘curly braces’ (“{“, “}”) (see “***psiaCommonTypes.xsd***”).
- **Raw strings**: All raw strings are encapsulated in ASCII double quotes (“) to allow the inclusion of non-standard characters with a standard delimiter.

Please note that all of the standard W3C rules apply regarding the replacement of non-allowable URI characters with their ASCII hex codes preceded by the percent symbol “%”. For example, a raw string with a ‘blank’ character would look like this: <URI>?lastID=”ONE%20MORE%20ITEM”.

11 Acknowledgements

This document and the PSIA protocol model would not have been possible without significant contributions by various member companies. While the efforts of all our members are appreciated, the PSIA would like to explicitly acknowledge the contributions of Cisco, Object Video, GE Security, Genetec, MileStone, Texas Instruments, IQInvision, Pelco, IBM, UTC Fire and Security, and Honeywell for their contributions of Intellectual Property, Market Requirements and technical activity.

12 PSIA XML Namespace Conventions

12.5 Root

The PSIA XML Namespace root is:

http://www.psialliance.org/schemas

All of the XML Schema Documents for PSIA protocols are published in a tree-structured hierarchy descending from this root.

12.6 Functional Level

The first level of branching within this hierarchy supports differentiation of documents based on function. There is a separate branch to contain common service documents and a branch for each of the respective protocols.

The functional branches include:

- System – contains all common service documents this includes the Common Metadata and Event Model (CMEM) and the Common Security (CSEC) Model.
- IPMD – IP Media Device
- RaCM – Recording and Content Management
- Analytics – Video Analytics
- AreaControl – Area Control

12.7 Version Level

The second level of branching supports differentiation of documents based on version. Within each functional branch, a separate branch will be created for each version. The version nodes contain the actual schema documents.

12.8 Versioning and References

In order to support flexibility in the development of the respective protocols and common services, each PSIA protocol document and service model document may reference schemas from different functional areas.

At version level 1.0 and 1.1, each document that references another schema MAY contain a section specifying which schema documents it references using the URI's of these referenced documents.

As of version level 1.2 of each document that references another schema MUST contain a section specifying which schema documents it references using the URI's of these referenced documents.

12.9 Enumeration of Documents

The URI of the PSIA Schema repository is:

- http://www.psialliance.org/schemas/

All active and current XSD files are listed, by specification category, and version, on this web page.

13 Version Management of Functional APIs

Each Working Group within PSIA is responsible for their own respective technical domains. This means that specifications, using the architectural and design standards contained herein, will be issued by each Working Group defining the functional characteristics of a ‘resource set’. A ‘resource set; is comprised of one, or more, REST resources that typically contain a set of operational parameters represented by XML documents. The REST resources and their respective schema documents are called ‘resource objects’. A set of Resource Objects comprise the equivalent of an API. Since APIs progress, as

time goes forward, the following Version Management guidelines apply to the creation, detection, and use of API/Resource Objects:

- All Working Groups are to create Resource Object sets that are backwards compatible, flexible, and forwardly extensible as, much as is technically possible.
- ALL PSIA schemas MUST bear version attribution in order to clearly identify which version of a document/parameter set is being employed, It is highly recommended that all critical elements within a schema also bear version attribution; especially those critical elements that are enumerations.
- ALL PSIA nodes MUST implement the '/PSIA/profile' resource thus declaring all PSIA specifications implemented by a node AND the associated spec version levels.
- If a Working Group determines that a new API (Resource Object set) definition can no longer be backwards compatible with the prior version, the following rules apply:
    - The Working Group MUST issue, in the current spec, the backwards support requirement for the deprecated API. I.e. the Working Group MUST provide a migration path from the older API to the newer, incompatible one. An older API may NOT be made obsolete immediately, but a finite time frame may be specified by the Working Group such that all client, management, system and device entities may know exactly what the migration plans and timeframes are, and what the backwards support liability is.
    - Where an API, or REST resource hierarchy, functionally deviates from an older API in an incompatible fashion,, the REST resource URIs MUST remain compatible up to the point of where the functional compatibility between the resource objects diverges. At the point in the REST resource hierarchy the APIs diverge, a 'version tag' MUST be inserted into the subordinate REST resource objects. For example, if the CMEM 'Actions' API/.Resource Object set was modified, in a 'Version 2' (for example), such that it was no longer compatible, the URI of the modified resource objects would be listed behind a "/v2" tag in the URI. Examples follow…

      ```
      /PSIA/Metadate/Actions/…    (Original API)
      /PSIA/Metadata/Actions/v2/… (Incompatible Version 2 API)
      ```

      OR….

      ```
      /PSIA/Metadate/Actions/…     (Original API)
      /PSIA/Metadata/Actions/v2.1/…(Incompatible Version 2.1 API)
      ```

      Please note that every effort should be made to preserve API compatibility between specification versions. And, in the event compatibility must be broken, version identification in the REST resource hierarchy must follow the formats depicted above. Additionally, the other Version Management requirements (i.e. schema versioning, etc.) still apply.

14 Appendices

14.1 Schemas

The following data structures are defined for use with the PSIA Service Model The format used in this section are basic samples intended to quickly demonstrate the structure of the data blocks. Note that the actual PSIA protocols are to include their documented data structures as .xsd files.

14.1.1 ResourceDescription

```
<ResourceDescription version="1.0" xmlns="urn:psialliance-org:resourcedescription">
    <name>index</name>
    <version>1.0</version>
    <type>resource</type>
    <get>
            <queryStringParameterList>none</queryStringParameterList>
            <inboundXML>none</inboundXML>
            <function>enumerate 1st level children</function>
            <returnResult>ResourceList</returnResult>
            <notes>non-recursive</notes>
    </get>
    <put></put>
    <post></post>
    <delete></delete>
</resourceDescription>
```

14.1.2 ResourceList

```
  <?xml version="1.0" encoding="utf-8" ?>
- <ResourceList version="1.0" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns="urn:psialliance-org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:psialliance-org http://www.psialliance.org/XMLSchemas/service.xsd">
-   <Resource version="1.0" xmlns="urn:psialliance-org" xlink:href="/index">
            <name>index</name>
            <type>resource</type>
  </Resource>
- <Resource xlink:href="/System">
    <name>System</name>
    <type>service</type>
-   <ResourceList>
-           <Resource xlink:href="/System/Network">
                    <name>Network</name>
                    <type>service</type>
-                   <ResourceList>
-                           <Resource xlink:href="/System/Network/ipAddress">
                                    <name>ipAddress</name>
                                    <type>resource</type>
                            </Resource>
                    </ResourceList>
            </Resource>
    </ResourceList>
  </Resource>
  </ResourceList>
```

14.1.3 QueryStringParameterList

```
  <?xml version="1.0" encoding="utf-8" ?>
- <QueryStringParameterList version="1.0" xmlns="urn:psialliance-org">
-   <QueryStringParameter>
          <name>positionX</name>
          <type>integer</type>
          <description>X position of scaling window</description>
    </QueryStringParameter>
  </QueryStringParameterList>
```

14.1.4 responseStatus

```
  <?xml version="1.0" encoding="utf-8" ?>
- <ResponseStatus version="1.0" xmlns="urn:psialliance-org">
  <requestURL>/Streaming/Channels</requestURL>
  <statusCode>1</statusCode>
    <!-- O=1-OK, 2-Device Busy, 3-Device Error, 4-Invalid Operation, 5-Invalid XML Format, 6-
Invalid XML Content; 7-Reboot Required-->
  <statusString>OK</statusString>
  <ID>1</ID>
  </ResponseStatus>
```

14.1.5 Service.xsd

The following XML Schema Document contains XML schema definitions for all of the PSIA Service Model data structures. All PSIA specifications are to use this schema document to maintain consistency of the PSIA Service Model data structures.

This document and all subsequent PSIA XML Schema Documents will be posted at http://www.psialliance.org/XMLSchemas.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema version="2.0"
           targetNamespace="urn:psialliance-org"
           xmlns="urn:psialliance-org"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xlink="http://www.w3.org/1999/xlink"
                  elementFormDefault="qualified">
  <xs:import namespace="http://www.w3.org/1999/xlink" schemaLocation="xlink.xsd"/>

  <xs:annotation>
    <xs:documentation xml:lang="en">
      PSIA Core Service Schema
    </xs:documentation>
  </xs:annotation>

  <!-- ID -->
    <xs:annotation>
       <xs:documentation>
              THE FOLLOWING ID TYPE HAS BEEN DEPRECATED. ALL SERVICE MODEL
              V2.0, AND LATER, IMPLEMENTATIONS ARE TO USE THE "LocalID" TYPE
              IN 'PSIACOMMONTYPES.XSD' IN THEIR NEW SCHEMA DEFINITIONS
       </xs:documentation>
    </xs:annotation>
  <xs:simpleType name="Id">
    <xs:restriction base="xs:string">
      <!-- TODO -->
    </xs:restriction>
  </xs:simpleType>
```

<!-- StatusCode --> The action of updating a node's configuration data, 'en masse', may cause a node to either A) require a client initiated reboot, B) cause the node to reboot on its own, or C) cause no resetting at all. If a node needs a client to reboot it, the statusCode in the response status must be "7" (Reboot required). If a node is going to reboot on its own, the statusCode must be "2" (Device busy). Otherwise all other operations should render the appropriate statusCode.

```
  <xs:simpleType name="StatusCode">
    <xs:restriction base="xs:int">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="7"/>
    </xs:restriction>
       <!-- O=1-OK, 2-Device Busy, 3-Device Error, 4-Invalid Operation, 5-Invalid XML
Format, 6-Invalid XML Content; 7-Reboot Required-->
  </xs:simpleType>

  <!-- ResourceType -->
  <xs:simpleType name="ResourceType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="service"/>
      <xs:enumeration value="resource"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- QueryStringParameter -->
  <xs:complexType name="QueryStringParameter">
    <xs:sequence>
       <xs:element name="name" type="xs:string" />
       <xs:element name="type" type="xs:string" />
       <xs:element name="description" type="xs:string" minOccurs="0" maxOccurs="1"/>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

```
  <!-- QueryStringParameterList -->
  <xs:complexType name="QueryStringParameterList">
    <xs:sequence>
      <xs:element name="QueryStringParameter" type="QueryStringParameter" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <!-- URLParameters -->
  <xs:complexType name="URLParameters">
    <xs:sequence>
      <xs:element name="queryStringParameterList" type="QueryStringParameterList" />
         <xs:element name="inboundData" type="xs:string" />
         <xs:element name="returnResult" type="xs:string" />
         <xs:element name="function" type="xs:string" />
         <xs:element name="notes" type="xs:string" />
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

<!-- ResponseStatus -->
  <xs:complexType name="ResponseStatus">
    <xs:sequence>
      <xs:element name="requestURL" type="xs:anyURI" />
      <xs:element name="statusCode" type="StatusCode" />
      <xs:element name="statusString" type="xs:string" />
      <xs:element name="id" type="Id" minOccurs="0" maxOccurs="1" />
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
       <xs:attribute name="version" type="xs:string" use="required"/>
  </xs:complexType>

  <!-- ResourceDescription -->
  <xs:complexType name="ResourceDescription">
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
         <xs:element name="version" type="xs:string" />
         <xs:element name="type" type="ResourceType" />
         <xs:element name="description" type="xs:string" minOccurs="0" maxOccurs="1"/>
         <xs:element name="notes" type="xs:string" minOccurs="0" maxOccurs="1"/>
         <xs:element name="get" type="URLParameters" />
         <xs:element name="put" type="URLParameters" />
         <xs:element name="post" type="URLParameters" />
         <xs:element name="delete" type="URLParameters" />
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="version" type="xs:string" use="required"/>
  </xs:complexType>

  <!-- Resource -->
  <xs:complexType name="Resource">
    <xs:sequence>
      <xs:element name="name" type="xs:string" />
         <xs:element name="version" type="xs:string" />
         <xs:element name="type" type="ResourceType" />
         <xs:element name="description" type="xs:string" minOccurs="0" maxOccurs="1"/>
         <xs:element name="ResourceList" type="ResourceList" minOccurs="0" maxOccurs="1"/>
      <xs:any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="version" type="xs:string" use="required"/>
  </xs:complexType>
```

```
  <!-- ResourceList -->
  <xs:complexType name="ResourceList">
    <xs:sequence>
      <xs:element name="Resource" type="Resource" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="version" type="xs:string" use="required"/>
  </xs:complexType>

</xs:schema>
```

14.1.6 psiaCommonTypes.xsd

The following XSD schema file contains the 'types' (simple and complex types) that are common to most of the schemas published by the various PSIA working groups. This file MUST be consulted first for types that are commonly used in PSIA schema definitions and publishing.

```
<?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="1.0">

  <!-- ================================================================ -->
  <!-- ================= Extension Object ============================= -->
  <!-- ================================================================ -->
  <xs:complexType name="CommonTypeCustomExtension">
    <xs:sequence>
      <!-- For interoperability, name must be registered with PSIA + XSD provided for the any-
obj -->
      <xs:element name="CustomExtensionName" minOccurs="1" maxOccurs="1" type="xs:anyURI" />
      <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

<xs:schema targetNamespace="urn:psialliance-org" elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="urn:psialliance-org" version="1.0">

  <!-- ================================================================ -->
  <!-- ================= Extension Object ============================= -->
  <!-- ================================================================ -->
  <xs:complexType name="CommonTypeCustomExtension">
    <xs:sequence>
      <!-- For interoperability, name must be registered with PSIA + XSD provided for the any-
obj -->
      <xs:element name="CustomExtensionName" minOccurs="1" maxOccurs="1" type="xs:anyURI" />
      <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
  <!-- ================================================================ -->
  <!-- ================================================================ -->

  <!-- ================================================================ -->
  <!-- ================= ID Types ===================================== -->
  <!-- ================================================================ -->
  <xs:simpleType name="GlobalID">
    <xs:annotation>
      <xs:documentation xml:lang="en">
        The representation of a GUID, generally the 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="ReferenceID">
    <xs:sequence>
```

```
      <xs:element name="ID" minOccurs="1" maxOccurs="1" type="LocalID" />
      <xs:element name="GUID" minOccurs="0" maxOccurs="1" type="GlobalID" />
      <xs:element name="Name" minOccurs="0" maxOccurs="1" type="xs:string" />
      <xs:element name="CustomExtension" minOccurs="0" maxOccurs="unbounded"
type="CommonTypeCustomExtension" />
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>
  <!-- ================================================================ -->
  <!-- ================================================================ -->

  <!-- ================================================================ -->
  <!-- ================= Access Related ============================== -->
  <!-- ================================================================ -->
  <xs:complexType name="PartitionMemberIDList">
    <xs:sequence>
      <xs:element name="PartitionMemberID" type="ReferenceID" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PartitionIDList">
    <xs:sequence>
      <xs:element name="PartitionID" type="ReferenceID" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="PortalIDList">
    <xs:sequence>
      <xs:element name="PortalID" type="ReferenceID" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="IdentifierInfoList">
    <xs:sequence>
      <xs:element name="IdentifierInfo" type="IdentifierInfo" minOccurs="1"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="IdentifierInfo">
    <xs:sequence>
      <xs:element name="Type" type="IdentifierType"/>
      <xs:element name="Value" type="xs:string">
        <xs:annotation>
          <xs:documentation>
            This should hold card detail such as card number in case
            the identifier is a card. If this is used to specify a user
            code/PIN or biometric info of a user, it must be encrypted.
            The encrytion must be as specified in Common Security (CSEC)
            specification. It is okay to return an empty string if the
            encrypted user code/PIN or biometric info can not be sent
            over network due to security regulations.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="IdentifierType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Card"/>
      <xs:enumeration value="PIN"/>
      <xs:enumeration value="Biometric"/>
      <xs:enumeration value="KeyFob"/>
```

```
    </xs:restriction>
  </xs:simpleType>
  <!-- ================================================================ -->
  <!-- ================================================================ -->

  <!-- ================================================================ -->
  <!-- ================= Storage and Content Related ================== -->
  <!-- ================================================================ -->
  <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>
  <!-- ================================================================ -->
  <!-- ================================================================ -->

  <!-- ================================================================ -->
  <!-- ================= Time Related ================================= -->
  <!-- ================================================================ -->
  <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:complexType name="TimeScheduleIDList">
    <xs:sequence>
      <xs:element name="TimeScheduleID" type="ReferenceID" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="HolidayInfoList">
    <xs:sequence>
      <xs:element name="HolidayInfo" type="HolidayInfo" minOccurs="0" maxOccurs="unbounded"/>
```

```
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="HolidayInfo">
    <xs:sequence>
      <xs:element name="ID" type="LocalID"/>
      <xs:element name="UID" type="GlobalID" minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            A UID may be used when this definition should be
            shared across multiple systems.
          </xs:documentation>
        </xs:annotation>
      </xs:element>      <xs:element name="Name" type="xs:string" minOccurs="0"/>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
      <xs:element name="RecursYearly" type="xs:boolean">
        <xs:annotation>
          <xs:documentation>
            Does this holiday occur only on this date or recurs yearly?
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="StartDate" type="xs:date"/>
      <xs:element name="EndDate" type="xs:date"/>
      <xs:element name="CustomExtension" minOccurs="0" maxOccurs="unbounded"
type="CommonTypeCustomExtension" />
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="TimeScheduleInfoList">
    <xs:sequence>
      <xs:element name="TimeScheduleInfo" type="TimeScheduleInfo" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="TimeScheduleInfo">
    <xs:sequence>
      <xs:element name="ID" type="LocalID" />
      <xs:element name="UID" type="GlobalID" minOccurs="0">
        <xs:annotation>
          <xs:documentation>
            A UID may be used when this definition should be
            shared across multiple systems.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
      <xs:element name="TimeIntervalInfoList" type="TimeIntervalInfoList"/>
      <xs:element name="CustomExtension" minOccurs="0" maxOccurs="unbounded"
type="CommonTypeCustomExtension" />
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="TimeIntervalInfoList">
    <xs:sequence>
      <xs:element name="TimeIntervalInfo" type="TimeIntervalInfo" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="TimeIntervalInfo">
```

```
    <xs:sequence>
      <xs:element name="Day" type="Day"/>
      <xs:element minOccurs="0" name="Holiday" type="xs:integer">
        <xs:annotation>
          <xs:documentation>
            This is only required if Day is holiday and
            the time interval doesn't apply to all holidays.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="StartTime" type="xs:time"/>
      <xs:element name="EndTime" type="xs:time"/>
      <xs:element name="CustomExtension" minOccurs="0" maxOccurs="unbounded"
type="CommonTypeCustomExtension" />
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>

  <xs:simpleType name="Day">
    <xs:restriction base="xs:string">
      <xs:enumeration value="All"/>
      <xs:enumeration value="Sunday"/>
      <xs:enumeration value="Monday"/>
      <xs:enumeration value="Tuesday"/>
      <xs:enumeration value="Wednesday"/>
      <xs:enumeration value="Thursday"/>
      <xs:enumeration value="Friday"/>
      <xs:enumeration value="Saturday"/>
      <xs:enumeration value="Holiday"/>
    </xs:restriction>
  </xs:simpleType>
  <!-- ================================================================ -->
  <!-- ================================================================ -->
```

14.1.7 profile.xsd (for “/PSIA/profile”)

The following XSD schema file provides all of the definitions for the ‘profile’ advertised by all PSIA nodes.

```
<?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/1.1/psiaCommonTypes.xsd"/>

<xs:element name="PsiaProfile" version="1.1" minOccurs="1" maxOccurs="1" type="PsiaProfile"/>

<xs:complexType name="PsiaProfile">
    <xs:sequence>
            <xs:element name="systemID" minOccurs="1" maxOccurs="1" type="GlobalID"/>
            <xs:element name="nativeID" minOccurs="1" maxOccurs="1" type="GlobalID"/>
            <xs:element name="psiaServiceVersion" minOccurs="1" maxOccurs="1" type="xs:float"/>
            <xs:element name="primaryPsiaSpec" minOccurs="1" maxOccurs="1"
type="PsiaSpecDecl"/>
            <xs:element name="otherSpecList" minOccurs="0" maxOccurs="1" type="PsiaSpecList"/>
            <xs:element name="profileList" minOccurs="0" maxOccurs="1" type="PsiaProfileList"/>
            <xs:element name="nodeDescription" minOccurs="0" maxOccurs="1" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="PsiaSpecDecl">
    <xs:sequence>
            <xs:element name="psiaSpecName" minOccurs="1" maxOccurs="1" type="PsiaSpecTag"/>
            <xs:element name="psiaSpecVersion" minOccurs="1" maxOccurs="1" type="xs:float"/>
            <xs:element name="psiaSpecProfile" minOccurs="1" maxOccurs="1"
type="PsiaSpecProfileLevel"/>
            <xs:element name="psiaSpecInfo" minOccurs="0" maxOccurs="1" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="PsiaProfileDecl">
    <xs:sequence>
            <!-- profile name is string for now  -->
            <xs:element name="psiaProfileName" minOccurs="1" maxOccurs="1" type="xs:string"/>
            <xs:element name="psiaProfileVersion" minOccurs="1" maxOccurs="1" type="xs:float"/>
            <xs:element name="psiaSpec" minOccurs="1" maxOccurs="1" type="PsiaSpecTag"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="PsiaProfileList">
    <xs:sequence>
            <xs:element name="psiaProfileDefn" minOccurs="1" maxOccurs="unbounded"
type="PsiaProfileDecl"/>
    </xs:sequence>
</xs:complexType>


<xs:simpleType name="PsiaSpecTag">
    <xs:restriction base="xs:string">
            <xs:enumeration value="ipmd"/>
            <xs:enumeration value="racm"/>
            <xs:enumeration value="videoAnalytics"/>
            <xs:enumeration value="cmem"/>
            <xs:enumeration value="areaCtl"/>
            <xs:enumeration value="csec"/>
            <xs:enumeration value="other-PSIA"/>
            <xs:enumeration value="other-private"/>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="PsiaSpecList">
    <xs:sequence>
            <xs:element name="psiaSpecDefn" minOccurs="1" maxOccurs="unbounded"
type="PsiaSpecDecl"/>
```

```
    </xs:sequence>
</xs:complexType>

<xs:simpleType name="PsiaSpecProfileLevel">
    <xs:restriction base="xs:string">
            <xs:enumeration value="core"/>
            <xs:enumeration value="core+"/>
            <xs:enumeration value="basic"/>
            <xs:enumeration value="basic+"/>
            <xs:enumeration value="extnded"/>
            <xs:enumeration value="extended+"/>
            <xs:enumeration value="full"/>
            <xs:enumeration value="full+"/>
            <xs:enumeration value="advanced"/>
            <xs:enumeration value="advanced+"/>
    </xs:restriction>
</xs:simpleType>

</xs:schema>
```

14.1.8 batteryEvents.xsd (for /System/Battery related state events)

```
<?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/events/v1.1/metaHeader.xsd" />

>xs:element name="MetaHeader" minOccurs="1" maxOccurs="1" type="metaHeader"/>
<xs:element name="BatteryEvent" type="BatteryEvent"/>

<xs:complexType name="BatteryEvent">
    <xs:sequence>
                <xs:element name="id" minOccurs="1" maxOccurs="1" type="xs:integer"/>
                <xs:element name="state" minOccurs="1" maxOccurs="1" type="BatteryState"/>
                <xs:element name="event" minOccurs="1" maxOccurs="1" type="BatteryEventType"/>
                <xs:element name="chargeLevel" minOccurs="0" maxOccurs="1"
type="BatteryChargeLevel"/>
        </xs:sequence>
</xs:complexType>

<xs:simpleType name="BatteryState">
        <xs:restriction base="xs:string:">
                <xs:enumeration value=""active-primary"/>
                <xs:enumeration value="active-alternate"/>
                <xs:enumeration value="inactive-standby"/>
                <xs:enumeration value="absent"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="BatteryEventType">
        <xs:restriction base="xs:string:">
                <xs:enumeration value="state"/>
                <xs:enumeration value="charge"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="BatteryChargeLevel">
```

```
        <xs:sequence>
                <xs:element name="threshold" minOccurs="1" maxOccurs="1"
type="BatteryThreshold"/>
                <xs:element name="indicationState" minOccurs="1" maxOccurs="1"
type="BatteryChargeState"/>
                <xs:element name="indicationType" minOccurs="1" maxOccurs="1"
type="BatteryChargeType"/>
                <xs:element name="indicationLevel" minOccurs="1" maxOccurs="1"
type="xs:float"/>
        </xs:sequence>
</xs:simpleType>

<xs:simpleType name="BatteryTheshold">
        <xs:restriction base="xs:string:">
                <xs:enumeration value="low"/>
                <xs:enumeration value="critical"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="BatteryChargeState">
        <xs:restriction base="xs:string:">
                <xs:enumeration value="increasing"/>
                <xs:enumeration value="decreasing"/>
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="BatteryChargeType">
        <xs:restriction base="xs:string:">
           <xs:enumeration value=”percent”/>
           <xs:enumeration value="voltage"/>
        </xs:restriction>
</xs:simpleType>

</xs:schema>\>
```