A Proposed Application Programming Interface for a Physical Volume Repository

Merritt Jones
The MITRE Corporation
1820 Dolley Madison Blvd.
McLean VA 22102
MERRITT@mitre.org (Merritt E Jones)
Tel: 703-883-5471
Fax: 703-883-5230

Joel Williams
Systems Engineering and Security
7474 Greenway Center Drive
Greenbelt MD 20770
joel.williams@ses-inc.com (Joel Willams)
Tel: 301-441-3694
Fax: 301-441-3697

Richard Wrenn
Digital Equipment Corporation
Colorado Springs CO 80919
wrenn@cookie.enet.dec.com (Rich Wrenn)
Tel: 719-548-2887
Fax: 719-548-6330

Introduction

The IEEE Storage System Standards Working Group (SSSWG) has developed the *Reference Model for Open Storage Systems Interconnection, Mass Storage System Reference Model Version 5*. This document, dated September 8, 1994, provides the framework for a series of standards for application and user interfaces to open storage systems. More recently, the SSSWG has been developing Application Programming Interfaces (APIs) for the individual components defined by the model. The API for the Physical Volume Repository is the most fully developed, but work is being done on APIs for the Physical Volume Library and for the Mover also. The SSSWG meets every other month, and meetings are open to all interested parties. Further information on the SSSWG may be found at *http://www.arl.mil/IEEE/ssswg.html*.

The Physical Volume Repository (PVR) is responsible for managing the storage of removable media cartridges and for mounting and dismounting these cartridges onto drives. This document describes a model which defines a Physical Volume Repository, and gives a brief summary of the Application Programming Interface (API) which the IEEE Storage Systems Standards Working Group (SSSWG) is proposing as the standard interface for the PVR.

The IEEE Reference Model

The Reference Model is described in [1]. What follows is a very brief overview.

The Reference Model determines the following key definitions:

- *Store*: an addressable storage space, either physical or virtual
  - Physical attributes defined by the media type
  - Virtual attributes defined by the client request
- *Device*: A set of media access points (for data access) and mount points (for physical access.
- *Physical Volume*: The recording medium accessible without intervening load operations.
- *Cartridge*: A set of physical volumes or cartridges.

It also defines the following modules

- *Physical Volume Repository (PVR)*: It sees cartridges and drive mount points, and its major operation is to *mount* cartridges.
- *Physical Volume Library (PVL)*: It makes the volume to cartridge mapping and causes the PVR to mount cartridges. Its major operation is to *mount* physical volumes.
- *Virtual Storage Server (VSS)*: It creates virtual stores and performs the store to volume mapping. Its major operation is to create and manage virtual stores.
- *Mover (MVR)*: It manages data transfer and is designed in particular to manage high-speed data transfer. Its major operation is to *load* media to media access points and to perform data transfer.

Figure 1 depicts the IEEE Reference Model

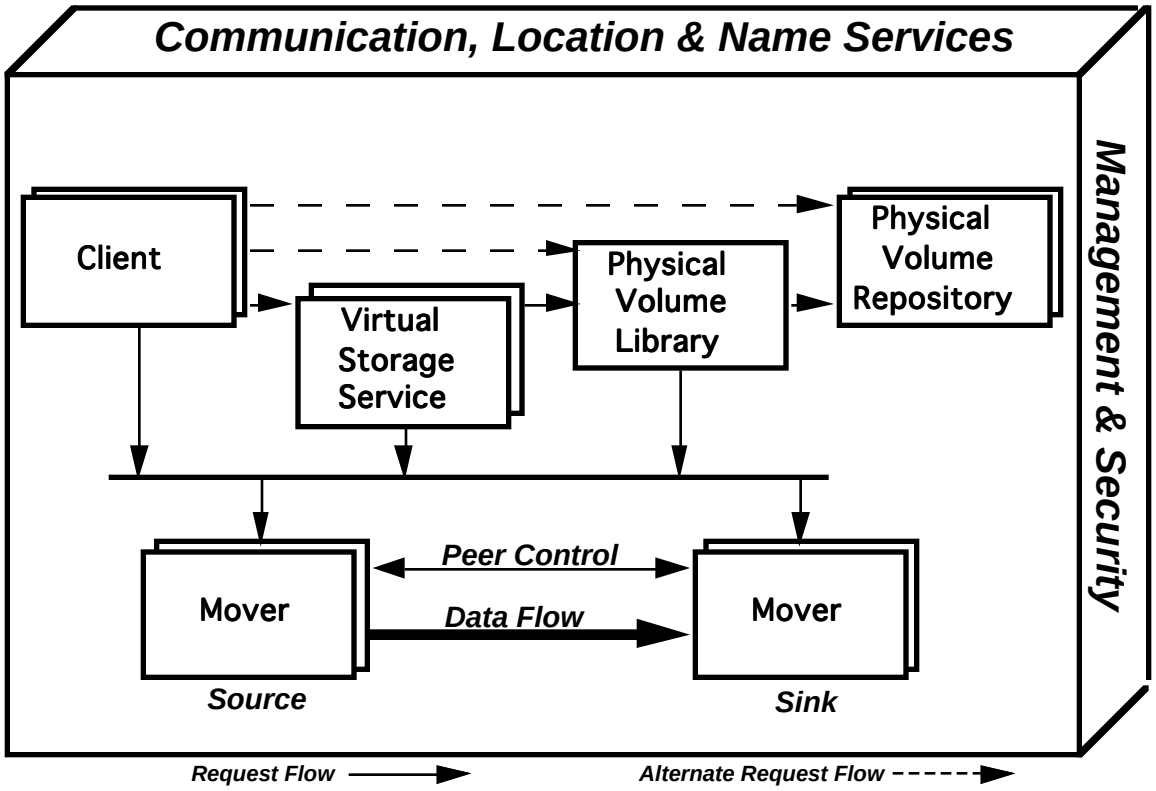

Figure 1

The PVR System

The PVR model and API define a consistent interface that client applications can use to interact with a variety of media-handling systems. The model and API that are described in this paper are based on a paper written by Rich Wrenn [2]. The PVR services are realized by servers which exist for each PVR within an enterprise. That is, the PVR server for various vendors' robotic systems, and humans performing manual operations, will be implemented uniquely. However, with a standard interface implemented, the interface presented to the client applications and the functions provided are exactly the same, regardless of the server implementation. According to this standards-based model, multiple PVR servers can exist on a single node, within a cluster, or within the enterprise. It is also possible for multiple client applications to share a PVR provided within the node, cluster, and enterprise.

Definition of a PVR

The physical volume repository is defined in the following way.

PVR Objects:

- One or more storage locations organized into partition objects
- Zero or more cartridge objects
- One or more media location domains for grouping cartridges

- Zero or more drive objects
- Zero or more device location domains for grouping drives or dependent PVRs
- Zero or more area objects for staging cartridges prior to mounts
- Zero or more port objects for ejecting cartridges from or injecting cartridges into the PVR
- One or more transfer mechanisms, either mechanical or human, operating in the same context, and capable of moving cartridges between their storage locations, drives, and ports
- One or more controllers which command the transfer mechanism within a single context

PVR Operations

- Mounting cartridges on drives and dismounting cartridges from drives
- Injecting and ejecting cartridges through ports. The PVR provides a controlled interface which defines directives that perform the above operations, and which allow object attributes to be accessed and modified.

Note that PVR does not access the drive data path and has no knowledge of the contents of cartridges.

Definition of the PVR Objects

Figure 1 depicts the relationships between the objects of the PVR. Child objects of the PVR are top level partitions, ports, drives, device-location domains, media-location domains, and cartridges. Partitions are arranged in a hierarchy, with the lowest level partition containing cartridges.

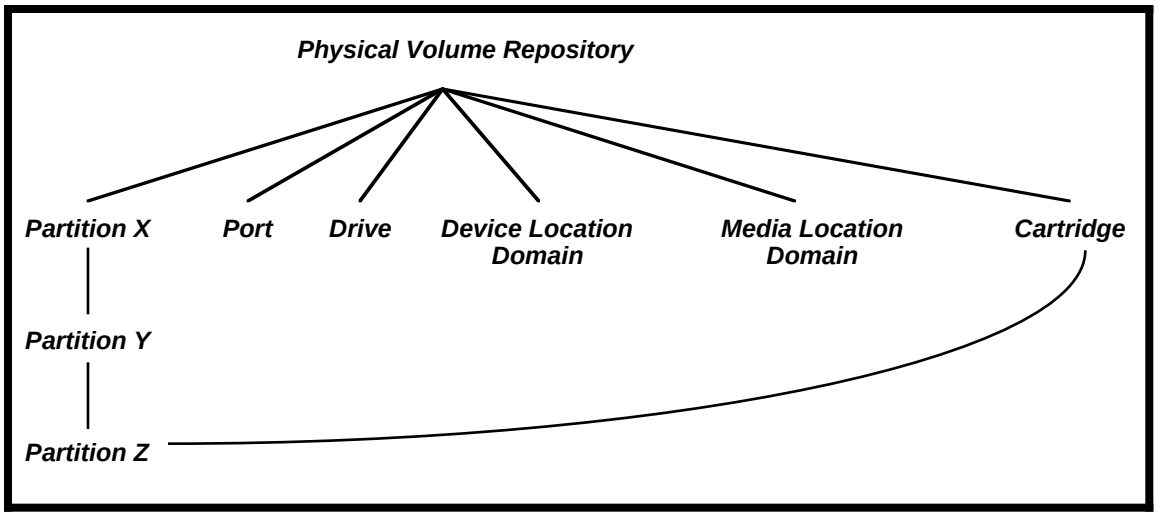

Figure 2

Magazine loaders are modeled through the concept of dependent and independent PVRs, so that a device into which cartridges may be mounted may be either a drive or a dependent PVR.

PVRs which are not serviced by other PVRs are defined as independent PVRs. An independent PVR is serviced by a tender who is responsible for transferring cartridges between the PVR's ports and the outside world. The tender must be able to remove cartridges which have been ejected through an outport and enter cartridges which have been injected through an inport.

PVRs which are serviced by other PVRs are defined as dependent PVRs. Dependent PVRs are not serviced by tenders. A dependent PVR has, by definition, one cartridge slot which can be mounted by another PVR. This cartridge slot is the location at which a compound cartridge (magazine) is mounted and dismounted.

The Partition Object

A partition describes the physical characteristics of a collection of media storage locations. It has finite capacity and stores media of exactly one cartridge type. The PVR cartridge storage locations are physically grouped and have equal characteristics with regard to location, cartridge type, and disaster protection. A top-level partition is a child of one, and only one PVR. Partitions may be created in a hierarchical fashion with a partition having a number of child partitions. A partition may contain one or more MLDs which may contain PVR cartridges.

The Media Location Domain (MLD)

The media location domain describes a logically grouped set of PVR cartridges with equivalent attributes located under a single top-level partition. It has finite capacity and stores PVR cartridges of exactly one type.

The Device Location Domain (DLD)

A device location domain (DLD) is a logical collection of PVR drives or dependent PVRs which have equal static attributes and may be used interchangeably. DLDs are characterized by connectivity, location, management policies, media types supported, and disaster protection. A DLD is a member of one and only one PVR.

The PVR Cartridge

The PVR cartridge is the PVR's view of the cartridge, which, as noted above, does not include any knowledge of the contents of the cartridge . Each cartridge resides in an MLD within a PVR and is referenced by PVR cartridge name. The PVR catalogs the exact physical location used to store the cartridge within a partition.

The PVR Drive

The drive is the PVR's view of the drive--essentially a place where it mounts and dismounts cartridges. A PVR drive name identifies the drives within a DLD. Each drive object resides in a DLD within a PVR. A PVR drive is a member of one and only one DLD.

The Area Object

An area is an optional, temporary location used to optimize mounting of cartridges into devices. Two types of areas are defined: stage areas and scratch areas. Cartridges are moved from their home location in the MLD to an area with either a stage or scratch directive in anticipation of a subsequent mount or mount scratch directive, respectively. This has the potential to reduce the apparent mount time of the PVR cartridge.

The Port Object

Ports are the physical portals through which cartridges are inserted and removed from PVRs. PVR ports are only associated with independent PVRs and are associated with globally named stations such that they are visible to external services, such as a transportation service. A single physical portal may function both as an inport and an outport.

The Task Object

PVR directives cause the instantiation of tasks. All tasks operate asynchronously from the thread which issued the directive, and there is a directive that allows for that thread to wait for the completion of the task. Directives are also provided which allow one to check on the status of a task or to cancel it.

Object Status

In general, each PVR object may be in one of three states:

- *Enabled*: The object is available for use.
- *Disabled*: The object is not available for use.
- *Error*: The object is in error.

Routines exist to *enable* or *disable* objects, and when an object is to be reconfigured or deleted, it must first be disabled.

Basic Directives

All objects (except cartridge and task objects) have four basic directives: *create*, *delete*, *set*, and *show*.

Create creates the object and specifies some of its attributes; ***Delete*** deletes the object, providing it is in the *disabled* state, and contains no references to other objects; ***Set*** changes or initializes a list of the object's attributes; ***Show*** displays a list of the object's attributes.

Cartridges have *set* and *show* directives, but no *create* and *delete* attributes. The *enter* and *inject* directives, which introduce a cartridge into its proper place in the PVR, correspond to the *create* directives for other objects, while the *eject* directive, which ejects the cartridge from the PVR, corresponds to the *delete* directives for other objects.

Tasks are created when a directive is issued which causes a task to begin.

Access Control

In the full implementation of the IEEE Reference Model, the primary clients of the PVR would be Storage Management and the Physical Volume Library.

The PVR server enforces access control through access control lists for all of its objects. The PVR, media location domain, device location domain, and task objects each have access control lists (ACL). Access control for each of the partition and PVR port objects is governed by the access control list on the PVR. Access control for each of the PVR cartridge and PVR area objects is governed by the access control list on the associated media location domain, and access control for each of the PVR drive objects is governed by the access control list on the associated device location domain.

Sharing the PVR by Non-Cooperating Clients

The PVR cartridges within an MLD have equivalent attributes, and are stored in locations such that all locations within the MLD yield equivalent cartridge attributes. Similarly, the devices within a DLD have equivalent attributes and reside in locations such that all locations within the DLD yield equivalent device attributes. MLDs, as well as DLDs, are mutually exclusive (non-overlapping).

The MLD and DLD concepts greatly simplify mount operations. In addition, the access lists on the MLDs and DLDs may be used to partition the PVR for use by non-cooperating clients. Each client would only see its collection of MLDs and DLDs, so that contention for drives and cartridges could not occur, and the possibility of deadlock is eliminated. The robot or human operator, however would be shared because it operates within a single context.

Access Control Rights

Within each of the access control lists, the following list of seven basic rights is used, with additional object specific rights defined as needed.

- *Show* is used for the sole purpose of limiting who can show the instance's attributes.
- *Set* is used primarily to control who can change or initialize attributes of instances governed by the access control list.
- *Control* is used for the sole purpose of limiting who can change the instance's access control list and other attributes (such as owner) which control access to the object.
- *Delete* is used primarily to control who can delete instances governed by the access control list.
- *Execute* is used for several purposes relating to action directives issued to the instance.
- *Read* is used to control the ability to read data contained in the instance.
- *Write* is used to control the ability to write data on the instance. It is also used to control who can change the membership of the instance and who can create subordinate objects of the instance.

Device Selection

The client applications can make three types of mount requests to the PVR:

- Explicit Device Selection (EDS): The client application directs the PVR to mount the PVR cartridge object onto a specific device.
- Automatic Device Selection (ADS): The client application directs the PVR to mount the PVR cartridge onto any device within a device location domain (DLD). The PVR selects a device and mounts the PVR cartridge onto that device. Finally, the PVR replies to the client application with the name of the device that it selected.
- Automatic Volume Recognition (AVR): The PVR is directed to mount the PVR cartridge onto any device within a device location domain (DLD) and the PVR selects a qualified device. The PVR does not return the name of the selected device to the client application. The client is required to determine the selected device through alternative means.

Populating PVRs with Cartridges

A PVR is populated with PVR cartridges in a two-step process. Either step may be performed first, but both steps must be completed before the PVR cartridge becomes available for use.

- *Inject* is a PVR directive issued by the PVR client which logically creates the PVR cartridge and assigns it to an MLD. The PVR cartridge name is provided by the client through the inject directive. It is not necessary for the PVR cartridge to be physically present for the inject to succeed.
  - If the PVR cartridge name is in the PVR and is in the *injected* or *available* state, reject the directive.
  - If the PVR cartridge name is in the PVR and is in the *entered* state, move the cartridge to the proper MLD and change the PVR cartridge state to *available.*
  - If the PVR cartridge is not in the PVR, create a PVR cartridge object placing it in the *injected* state. The PVR will then issue a request to a PVR tender that the PVR cartridge be physically *entered* through the PVR inport.
- *Enter* is an asynchronous action wherein a PVR cartridge is physically inserted into the PVR. The PVR cartridge name is provided when the cartridge is entered - manually or via a PVR vision system.
  - If the PVR cartridge name is in the PVR and is in the *entered* or *available* state, reject the cartridge.
  - If the PVR cartridge name is in the PVR and is in the *injected* state, move the cartridge to the proper MLD and change the PVR cartridge state to *available.*
  - If the PVR cartridge is not in the PVR, create a PVR cartridge object placing it in the *entered* state.

Object Attributes

Each object has four classes of attributes:

- *Identifier* attributes are used to name the object. Objects have a single identifier which is sometimes referred to as its primary identifier. Primary identifiers are specified during the creation of the object, or in the case of cartridges, when the cartridge is entered.
- *Characteristic* attributes describe the object and can be modified with the *set* directive. Some characteristic attributes must be specified when the object is created while others are optional and have default values. Frequently, the object must be in the *disabled* state in order for a characteristic attribute to be set.
- *Status* attributes describe the objects current status and its relationship to other objects cannot be modified with the set directive. In all cases, status attributes have default values.

- *Counter* attributes count events or actions. Initial values of counter attributes may be optionally specified on the object's *create* directive, but counter attributes cannot be modified with the set directive. Creation time is considered a counter attribute. On object creation, if creation time is not specified, it is set to the current time. For other counter attributes, if no initial value is specified, the attribute is set to zero during object creation.

Events

PVR objects generate events; in fact each counter attribute except creation time corresponds to an event. Typical events are *access denied* and *access granted.* A mechanism is provided to poll the PVR for a list of its events.

Characteristics of the Application Programming Interface

The Application Programming Interface which is being put forward as a proposed standard is a C-language interface which may be implemented in a distributed environment, such as the Distributed Computing Environment (DCE). References to objects are generally made through handles, which are opaque types containing information that allows the server to optimize access to the object. Each directive returns 0 on success and -1 on failure. In the case of failure, it sets the status variable to give further information (similar to the UNIX *errno*).

The following sections summarize the Application Programming Interface. It is fully documented in [3]. In addition to the directives listed below, there are convenience functions which free memory, handles which the server has allocated for the client, directives which list an object's members, and directives which enable or disable objects.

Handle Directives

The following routines take as input a name, or a name and another handle, and create a handle to the named object. Note that the task name is a 32-bit unsigned integer.

```
int pvr_area_get_handle (pvr_t pvr, wchar_t *area_name, pvr_area_t
   *area, pvr_status_t *status);

int pvr_dld_get_handle (pvr_t pvr, wchar_t *dld_name, pvr_dld_t *dld,
   pvr_status_t *status);

int pvr_drive_get_handle (pvr_t pvr, wchar_t *drive_name, pvr_drive_t
   *drive, pvr_status_t *status);

int pvr_get_handle (wchar_t *pvr_name, pvr_t *pvr_h, pvr_status_t
   *status);

int pvr_mld_get_handle (pvr_t pvr, wchar_t *mld_name, pvr_mld_t *mld,
   pvr_status_t *status);

int pvr_part_get_handle (pvr_t pvr, wchar_t *part_name, pvr_part_t
   *part, pvr_status_t *status);

int pvr_port_get_handle (pvr_t pvr, wchar_t *port_name, pvr_port_t
   *port, pvr_status_t *status);
```

```
int pvr_task_get_handle (pvr_t pvr, mss_unsigned32_t task_id,
   pvr_task_t *task, pvr_status_t *status);
```

Basic Create and Delete Directives

The following create and delete directives are available. The create directive supplies a linked list of attributes to be initialized upon creation. The error id identifies the first attribute, if any, that was in error.

```
int pvr_create (pvr_t pvr, pvr_id_attr_t *attribute_list, pvr_attr_t
   *error_id, pvr_status_t *status);
int pvr_delete (pvr_t pvr, pvr_status_t *status);

int pvr_area_create (pvr_area_t area, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_area_delete (pvr_area_t area, pvr_status_t *status);

int pvr_dld_create (pvr_dld_t dld, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_dld_delete (pvr_dld_t dld, pvr_status_t *status);

int pvr_drive_create (pvr_drive_t drive, pvr_id_attr_t
   *attribute_list, pvr_attr_t *error_id, pvr_status_t *status);
int pvr_drive_delete (pvr_drive_t drive, pvr_status_t *status);

int pvr_mld_create (pvr_mld_t mld, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_mld_delete (pvr_mld_t mld, pvr_status_t *status);

int pvr_port_create (pvr_port_t port, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_port_delete (pvr_port_t port, pvr_status_t *status);

int pvr_part_create (pvr_part_t part, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_part_delete (pvr_part_t part, pvr_status_t *status);

int pvr_task_delete (pvr_task_t task, pvr_status_t *status);
```

Basic Set and Show Directives

The set directives change or initialize the attributes in the linked list of attributes. Error identifies the first attribute, if any, that was in error. The show directive takes a linked list of attribute identifiers and creates a linked list giving the current state of the attributes of the object.

```
int pvr_set (pvr_t pvr, pvr_id_attr_t *attribute_list,  pvr_attr_t
   *error_id, pvr_status_t *status); int pvr_show (pvr_t pvr,
   pvr_attr_id_request_t *input_list,
pvr_id_attr_t **output_list , pvr_o_flag_t flag, pvr_status_t
   *status);
```

```
int pvr_area_set (pvr_area_t area, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_area_show (pvr_area_t area, pvr_attr_id_request_t
   *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
   pvr_status_t *status);

int pvr_cart_set (pvr_t pvr, wchar_t *name, pvr _id_attr_t
   *attribute_list, pvr_attr_t *error_id,
int pvr_cart_show (pvr_t pvr, wchar_t *name, pvr_attr_id_request_t
   *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
   pvr_status_t *status);

int pvr_cart_set (pvr_t pvr, wchar_t *name, pvr_id_attr_t
   *attribute_list, pvr_attr_t *error_id,
int pvr_cart_show (pvr_t pvr, wchar_t *name, pvr_attr_id_request_t
   *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
   pvr_status_t *status);

int pvr_drive_set (pvr_drive_t drive, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_drive_show (pvr_drive_t drive, pvr_attr_id_request_t
   *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
   pvr_status_t *status);

int pvr_mld_set (pvr_mld_t mld, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_mld_show (pvr_mld_t mld, pvr_attr_id_request_t *input_list,
   pvr_id_attr_t **output_list, pvr_o_flag_t *flag, pvr_status_t
   *status);

int pvr_part_set (pvr_part_t part, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_part_show (pvr_part_t part, pvr_attr_id_request_t
   *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
   pvr_status_t *status);

int pvr_port_set (pvr_port_t port, pvr_id_attr_t *attribute_list,
   pvr_attr_t *error_id, pvr_status_t *status);
int pvr_port_show (pvr_port_t port, pvr_attr_id_request_t
   *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
   pvr_status_t *status);

int pvr_task_set (pvr_task_t task, pvr_id_attr_t *attribute_list,
   pvr_status_t *status);
int pvr_task_show (pvr_task_t task, pvr_attr_id_request_t
   *input_list, pvr_id_attr_t **output_list, pvr_o_flag_t *flag,
   pvr_status_t *status);
```

Additional Area Directives

The *flush* directive returns all cartridges from the area to their home positions. The *stage* and *scratch* directives put the list of cartridges into the stage and scratch area respectively. The *list* directive lists the cartridges in an area. The *mount scratch* directive mounts from the scratch area.

```
int pvr_area_flush (pvr_area_t area, mss_unsigned32_t *task,
   pvr_status_t *status);
int pvr_area_stage (pvr_area_t area, pvr_string_list_t *cart_list,
   mss_unsigned32_t *task, pvr_status_t *status);
int pvr_area_scratch (pvr_area_t area, pvr_string_list_t *cart_list,
   mss_unsigned32_t *task, pvr_status_t *status);
int pvr_area_list (pvr_area_t area, pvr_string_list_t  **output_list,
   pvr_status_t *status);
int pvr_area_mount_scratch (pvr_area_t area, pvr_mount_type_t type,
   wchar_t *device_spec, wchar_t *info, mss_unsigned32_t *task,
   pvr_status_t *status);
```

Additional Cartridge Directives

The *enter* and *inject* directives place the cartridge into the PVR. The *eject* sends it out of the PVR. The *relocate* directive places the cartridge in a different location. There are different kinds of *mount* directives, and one *dismount* directive.

```
int pvr_cart_eject (pvr_t pvr, wchar_t *cart_name, wchar_t *port,
   mss_unsigned16_t *cell, pvr_status_t *status);

int pvr_cart_enter (pvr_t pvr, wchar_t *cart_name, wchar_t *port,
   pvr_status_t *status);

int pvr_cart_inject (pvr_t pvr, wchar_t *cart_name, wchar_t *port,
   mss_unsigned16_t *cell, wchar_t *mld, pvr_status_t *status);

int pvr_cart_relocate (pvr_t pvr, wchar_t *cart_name, wchar_t *mld,
   pvr_status_t *status);

int pvr_cart_bind_mount (pvr_t pvr, wchar_t *cart_name,
   pvr_comp_cart_list_t *list, pvr_mount_type_t type, wchar_t
   *device, mss_enum_boolean_t read_only, wchar_t *info,
   mss_unsigned32_t *task,  pvr_status_t *status);

int pvr_cart_bind_mount_scratch (pvr_t pvr, mss_unsigned16_t
   quantity, wchar_t *area, pvr_mount_type_t type, wchar_t *device,
   wchar_t *info, mss_unsigned32_t *task, pvr_status_t *status);

int pvr_cart_mount (pvr_t pvr, wchar_t *cart_name, mss_unsigned16_t
   side, pvr_mount_type_t  type, wchar_t *device, mss_enum_boolean_t
   read_only, wchar_t *info, mss_unsigned32_t *task,  pvr_status_t
   *status);

int pvr_cart_dismount (pvr_t pvr, wchar_t *cart_name, wchar_t *area,
   mss_unsigned32_t *task, pvr_status_t *status);
```

Additional PVR Directives

The pvr_ack_mount acknowledges that the client has detected that a mount or mount scratch operation has completed. The client provides the name of the device and cartridge that has been mounted

```
int pvr_ack_mount (pvr_task_t task, wchar_t *device_name, wchar_t
   *cart_name, mss_unsigned16_t *side, mss_enum_boolean_t process_check,
   pvr_status_t *status);
```

The pvr_update_mount_status function is sent to a dependent PVR to update the cartridge name that was mounted.

```
int pvr_update_mount_status (pvr_t pvr, wchar_t *mounted_cart_name,
   pvr_status_t *status);
```

Additional Drive Directive

The following directive enables the client to configure a drive.

```
int pvr_drive_configure (pvr_drive_t drive, wchar_t *node_name, wchar_t
   *local_drive_name, mss_enum_boolean_t *excluded_flag,  pvr_status_t
   *status);
```

Additional MLD Directive

The following directive lists the cartridges within an MLD.

```
int pvr_mld_list (pvr_mld_t mld, pvr_cart_state_t *state,
   pvr_slot_list_t **carts, pvr_status_t *status);
```

Additional Partition Directives

The pvr_part_copy directive streamlines configuration of partitions by allowing one to be a copy of another. The pvr_part_inventory directive starts a task which inventories the cartridges of a partition. The pvr_part_list directive provides a list of the cartridges in a partition.

```
int pvr_part_copy (pvr_part_t part, pvr_id_attr_t *attribute_list,
   mss_enum_boolean_t descendent_flag, wchar_t *old_part_spec,
   pvr_status_t *status);

int pvr_part_inventory (pvr_part_t part, mss_unsigned32_t start,
   mss_unsigned32_t count, mss_unsigned32_t *task, pvr_status_t
   *status);

int pvr_part_list (pvr_part_t part, pvr_cart_state_t *state,
   pvr_slot_list_t **carts, pvr_status_t *status);
```

Additional Task Directives

Directives are provided to manage tasks within the PVR. Tasks execute asynchronously, and the pvr_task_wait function allows the calling thread to wait for the task to complete. The pvr_task_cancel, pvr_task_pause, and pvr_task_resume allow for task management.

```
int pvr_task_cancel (pvr_task_t task, pvr_reason_code_t code,
   pvr_status_t *status);
int pvr_task_pause (pvr_task_t task, pvr_status_t *status);
int pvr_task_resume (pvr_task_t task, pvr_status_t *status);
int pvr_task_wait (pvr_task_t task, pvr_task_completion_status_t
   *task_status,  pvr_status_t *status);
```

Polling Directives

There is a collection of polling directives which allows the client to poll one instance or all instances of an object for a specified list of events.

```
int pvr_poll_events (pvr_t pvr, time_t start_time, pvr_event_request_t
   *input_list, pvr_id_event_t **output_list,  pvr_status_t *status);
int pvr_drive_poll_events (pvr_drive_t drive, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_all_drive_poll_events (pvr_t pvr, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_port_poll_events (pvr_port_t port, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_all_port_poll_events (pvr_t pvr, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_area_poll_events (pvr_area_t area, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_all_area_poll_events (pvr_t pvr, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_task_poll_events (pvr_task_t task, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_all_task_poll_events (pvr_t pvr, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_dld_poll_events (pvr_dld_t dld, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_all_dld_poll_events (pvr_t pvr, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
```

```
int pvr_dld_poll_events (pvr_dld_t dld, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_all_dld_poll_events (pvr_t pvr, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_mld_poll_events (pvr_mld_t mld, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
int pvr_all_mld_poll_events (pvr_t pvr, time_t start_time,
   pvr_event_request_t *input_list, pvr_id_event_t **output_list,
   pvr_status_t *status);
```

References

Links to the following documents may be found at *http://www.arl.mil/IEEE/ssswg.html.*

1. *Reference Model for Open Storage Systems Interconnection*, IEEE Storage System Standards Working Group, Project 1244, September 8, 1994.

2. *IEEE Storage System Standards Physical Volume Repository Model (proposed)*

3. *The Physical Volume Application Programming Interface*