Resource Functions I.

High Availability and Timeliness in Linux
Katalin Réti
02.04.2003

Introduction

The Hardware Platform Interface (HPI) provides a platform-independent interface to platform-specific management services. The components of the system are identified and described to the extent that they may be monitored, controlled, or detected by the platform management infrastructure of the system. The components are presented by the HPI in terms of how they are seen by the management infrastructure.

A component of the system is called an entity. Each entity is identified by an entity path, which is determined by the component's location in the physical containment hierarchy of the system.

Access to the entities in the system is through management access points in the system infrastructure, represented in the HPI as resources. Every resource is responsible for managing and presenting to the HPI user the entities that it has management control over.

The HPI view of a system is divided into one or more domains, where a domain provides access to some set of the resources within the system. A domain represents some part of the system that is capable of being managed by a single HPI user.

All access to the HPI is via a session. Each session is opened on a single domain. A session provides access only to resources that are visible in the domain upon which the session is opened.

The Resource Presence Table (RPT) contains an entry for each resource currently present in the domain. The RPT is automatically built and maintained by the HPI implementation. The RPT entry for a resource includes a flag that indicates what capabilities the resource supports.

Management Instruments

Management capabilities of system entities are modeled in the HPI with “management instruments”. There are four types of management instruments: sensors, controls, entity inventory repositories and watchdog timers. Management instruments are contained in HPI resources, and are associated with specific physical entities.

Entity inventory repository

Each physical entity in the system may have inventory data associated with it. This inventory data, such as manufacturer, model number, revision level, serial number, and static configuration information is accessible by reading records from an entity inventory repository. The HPI user may read the inventory data from any repository, and may also update the data in a repository.

Sensors

A sensor is used to monitor physical characteristics of an entity, such as temperature, voltages, latch positions, etc. Generally, sensors are used to model values that may change over time. They may report analog or discrete values.

Sensors also have associated event states. When changes in the reading occur, the sensor can assert or de-assert one or more of its event states. When a sensor asserts or de-asserts an event state, an HPI event may be created and sent to an HPI user or put in a log.

Controls

A control is used to send a command to an entity. There are six types of controls to handle different sorts of data that may need to be sent to an entity: digital, discrete, analog, stream, text and OEM defined.

Watchdog timers

A watchdog timer management instrument is used to control physical watchdog timers that may be implemented on physical entities. Specialized function calls are available to configure and start a watchdog timer, to send “keep-alive” heartbeats to it, and to define what actions are taken when it expires.

Resource Data Record Repository Management

Resource functions are used to access the Resource Data Record (RDR) repository for a specific resource. Every resource must have an associated RDR. The RDR repository holds information indicating the set of sensors, controls, watchdogs and entity inventory repositories for all of the entities that are managed by a resource. All sensors, controls, watchdogs and entity inventories present in a resource must be specified in the resource's RDR repository.

The concept of RDRs provides for much of HPI's portability and extensibility across a multitude of hardware platform implementations. Because each platform will have a different population of domain controllers and resources, the RDR concept provides a means of discovering and managing these varied populations of hardware platforms and sensors. The RDR repository is used during discovery to learn the management capabilities of the resource.

The HPI model uses a distributed repository where each resource maintains a local repository of records. At the resource level, the RDR repository is a logical database containing a collection of records that describe sensors, controls, watchdogs and entity inventory repositories. Each RDR contains common fields that define record type and naming information.

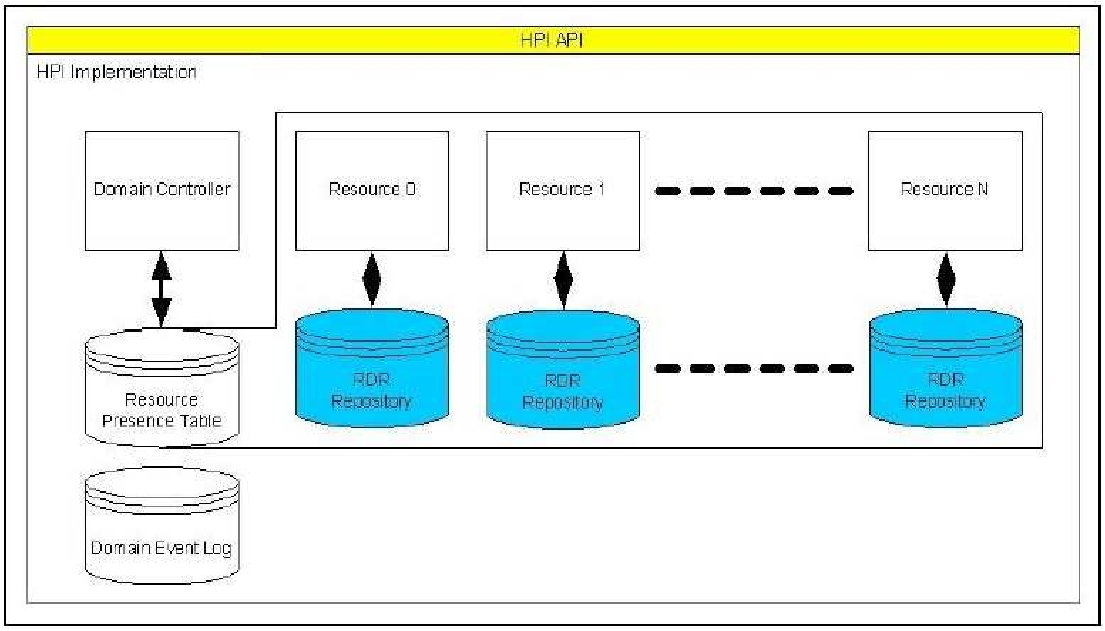

Fig. 1. Distributed RDR Repositories

Function **saHpiRdrGet()** returns a resource data record from the addressed resource.

Protoype

```
SaErrorT SAHPI_API saHpiRdrGet(
        SAHPI_IN SaHpiSessionIdT     SessionId,
        SAHPI_IN SaHpiResourceIdT    ResourceId,
        SAHPI_IN SaHpiEntryIdT       EntryId,
        SAHPI_OUT SaHpiEntryIdT      *NextEntryId,
        SAHPI_OUT SaHpiRdrT          *Rdr
);
```

Parameters

SessionId – [in] Handle to session context.

ResourceId – [in] Resource ID of the addressed resource.

EntryId – [in] Handle of the RDR to retrieve. Reserved entry ID values: SAHPI_FIRST_ENTRY Get first entry

SAHPI_LAST_ENTRY Reserved as delimiter for end of list. Not a valid entry identifier.

NextEntryId – [out] Pointer to location to store Entry ID of next entry in RDR repository.

Rdr – [out] Pointer to the structure to receive the requested resource data record.

Return value

SA_OK is returned on successful completion; otherwise, an errorcode is returned.

Sensor Functions

These functions are valid for resources that have the "Sensor" capability set in their corresponding RPT (Resource Presence Table) entries.

Sensors contain several configuration parameters that may be set with the appropriate set sensor functions. Typically, these parameters are set to appropriate defaults by the HPI implementation. These set functions are available for management software to override these defaults.

When a resource is re-initialized, the HPI implementation may reset these parameters. If the resource that hosts the sensor supports parameter control, it may be possible to store the newly loaded parameter values in non-volatile storage so that the new settings will remain with the resource through removal or reinsertation.

Function **saHpiSensorReadingGet()** is used to retrieve a sensor reading.

Parameters

SessionId – [in] Handle to session context.

ResourceId – [in] Resource ID of the addressed resource.

SensorNum – [in] Sensor number for which the sensor reading is being retrieved.

Reading – [out] Pointer to a structure to receive sensor reading values.

Function **saHpiSensorReadingConvert()** converts between raw and interpreted sensor reading values.

Parameters

SessionId – [in] Handle to session context.

ResourceId – [in] Resource ID of the addressed resource.

SensorNum – [in] Sensor number for which reading is associated.
ReadingInput – [in] Pointer to the structure that contains raw or interpreted reading to be converted.
ConvertedReading – [out] Pointer to structure to hold converted reading.

Function **saHpiThresholdsGet()** retrieves the threshold for the given sensor.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the addressed resource.
SensorNum – [in] Sensor number for which threshold values are being retrieved.
SensorThresholds – [out] Pointer to returned sensor thresholds.

Function **saHpiSensorThresholdSet()** sets the specified thresholds for the given sensor.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the addressed resource.
SensorNum – [in] Sensor number for which threshold values are being set.
SensorThresholds – [out] Pointer to sensor thresholds values being set.

Function **saHpiSensorTypeGet()** retrieves the sensor type and event category for the specified sensor.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the addressed resource.
SensorNum – [in] Sensor number for which the type is being retrieved.
Type – [out] Pointer to returned enumerated sensor type for the specified sensor.
Category – [out] Pointer to location to receive the returned sensor event category.

As mentioned in section “Management Instruments” sensors have associated event states. Each event state belongs to a category. An event state can be asserted or deasserted. When an event state changes from asserted to deasserted (or from deasserted to asserted) a message can be generated.

Function **saHpiSensorEventEnablesGet()** provides the ability to get the disable or enable event message generation status for individual sensor events. The sensor event states are relative to the event category specifies by the sensor.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the addressed resource.
SensorNum – [in] Sensor number for which the event enable configuration is being requested.
Enables – [out] Pointer to the structure for returning sensor status and event enable information.

Function **saHpiSensor EventEnablesSet()** provides the ability to set the disable or enable event message generation status for individual sensor events.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the addressed resource.
SensorNum – [in] Sensor number of which the event enables are being set.
Enables – [in] Pointer to the structure containing the enabled status for each event.

Aggregate Resource Status

If a resource entry has the capability bit of SAHPI_CAPABILITY_AGGREGATE_STATUS set to 1 it means that the resource supports aggregate resource status. Aggregate resource status provides a session owner the capability to quickly derive information about the functional, thermal, and power state of the single or primary entity associated with that resource. If a resource manages multiple entities, the entity path in the RPT entry for the resource indicates which entity the aggregate status sensors are associated with. A resource, which supports aggregate resource status must provide the following types of sensors: SAHPI_OPERATIONAL, SAHPI_POWER_UNIT and SAHPI_TEMPERATURE.

The aggregate operational status sensor reflects the functional status of the entity. There are two event states, one indicates that the entity is performing its intended function, the other one indicates that it is not.

The aggregate power status sensor reflects the power status of the entity. The power status is reported in terms of how it effects the operation of the resource. Six event states are used. Three of them signals a change to a degraded power status at the lower end of the range. The first one does not impact the functional status, the second one signals a change to a severely degraded power status that may impact the functional status, and the third one signals a change to a catastrophic power status that has impacted the functional status. The other three states signal a change to a degraded power status at the higher end of the range the same way as the above ones do on the lower end.

The aggregate thermal status sensor reflects the thermal status of the entity. The thermal status is reported in terms of how it effects the operation of the entity. Six event states are used. Three of them signals a change to a degraded thermal status at the lower end of the range. The first one does not impact the functional status, the second one signals a change to a severely degraded thermal status that may impact the functional status, and the third one signals a change to a catastrophic thermal status that has impacted the functional status. The other three states signal a change to a degraded power status at the higher end of the range the same way as the above ones do on the lower end.

Controls

These functions are valid for resources, that have the "Control" capability set in their corresponding RPT entries.

Function **saHpiControlTypeGet()** retrieves the type of a control object.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the addressed resource.
CtrlNum – [in] Control number.
Type – [out] Pointer to saHpiCtrlTypeT variable to receive the enumerated control type for the specified control.

Function **saHpiControlStateGet()** retrieves the current state of a control object.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] ResourceId of the addressed resource.
CtrlNum – [in] Number of the control for which the state is

being retrieved.
CtrlState – [in/out] Pointer to a control data structure into which the current control state will be placed.

Function **saHpiControlStateSet()** is used for setting the state of the specified control object.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the addressed resource.
CtrlNum – [in] Number of the control for which the state is being set.
CtrlState – [in/out] Pointer to a control state data structure holding the state to be set.

Entity Inventory Data

These functions are valid for resources that have the "Entity Inventory" capability set in their corresponding RPT entries. Resources that support entity inventory data may contain inventory data for for one or more entity inventory repositories. Each set of inventory data provides detailed identification information for a physical entity. The entities for which the resource contains inventory data may be found via Inventory Data Records in the RDR repository.

Inventory information is stored in an entity inventory repository in a set of variable length records. Functions are provided to read the whole inventory information from a particular entity inventory repository, or the write the whole inventory information in a particular repository. Except at initial manufacture, there will generally always be one or more inventory records already present in a repository, and management middleware will, at most, be adding to or modifying the existing inventory information. Therefore the typical usage is to first read the inventory data, and then modify it appropriately and then write the inventory data back to the entity inventory repository, as modified.

Function **saHpiEntityInventoryDataRead()** returns inventory data for a particular entity associated with a resource.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the addressed resource.
EirId – [in] Identifier for the entity inventory repository.
BufferSize – [in] Size of the InventData buffer passed in.

InventData – [in] Pointer to the buffer for the returned data.
ActualSize – [out] Pointer to size of the actual amount of data returned.

Function **saHpiEntityInventoryDataWrite()** writes the specified data to the inventory information area.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the addressed resource.
EirId – [in] Identifier for the entity inventory repository.
InventData – [in] Pointer to data to write to the repository.

Watchdog Timer

Many high availability platforms contain watchdog timers to provide a means of monitoring the overall health of the software system. The HPI provides a standardized set of functions to access these watchdog timers. Additionally, the HPI provides a means to configure selected actions that will be taken when the watchdog timer expires. These actions include power off, power cycle, and reset actions.

A timer use value is set whenever the watchdog is set. There are no restrictions on using timer use values, but some of them have pre-defined values.

The watchdog timer can be used as a two-stage watchdog. If the pre-timeout interrupt is set an interrupt occurs at a configured interval of time before the rimer expires. The pre-timeout interrupt handler may implement a wide variety of actions. While the type of interrupt is configured, the caller is responsible for installing the appropriate interrupt handler before setting the watchdog configuration.

More than one watchdog timer may be supported by a resource. The watchdog records in the RDR repository identify which entity a particular watchdog timer is associated with.

When a watchdog times out, in addition to the configured action, an event may be generated. It is possible to configure the watchdog to take no actions other than the generation of the event.

These functions are valid for resources that have the watchdog timer capability set in their corresponding RPT entries.

Function **saHpiWatchdogTimerGet()** retrieves the current watchdog timer settings and configuration.
Parameters
SessionId – [in] Handle to session context.

ResourceId – [in] Resource ID of the resource, which contains the watchdog timer being addressed.
WatchdogNum – [in] The watchdog number that specifies the watchdog timer on a resource.
Watchdog – [out] Pointer to watchdog data structure.

Function **saHpiWatchdogTimerSet()** provides a method for initializing the watchdog timer configuration.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the resource, which contains the watchdog timer being addressed.
WatchdogNum – [in] The watchdog number specifying the specific watchdog timer on a resource.
Watchdog – [out] Pointer to watchdog data structure.

Function **saHpiWatchdogTimerReset()** provides a method to start or restart the watchdog timer from the initial countdown value.
Parameters
SessionId – [in] Handle to session context.
ResourceId – [in] Resource ID of the resource, which contains the watchdog timer being addressed.
WatchdogNum – [in] The watchdog number specifying the specific watchdog timer on a resource.

References

(1) Service Availability Forum Interface Specification 1.0.
(2) Service Availability Forum specification white paper