RFC0096 - Power Management

Confidential, Draft

3 Pages

Abstract

Defines the management of system and device power states within an OSGi Service Platform

0 Document Information

0.1 Table of Contents

0.2 Status

This document specifies how the system and device power states are handled within an OSGI Service Platform, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi Alliance.

0.3 Acknowledgement

0.4 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

```
Source code is shown in this typeface.
```

0.5 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
0.1	Sep 01 2004	Document Creation
0.2	Sep 24 2004	• Change DevicePowerStateListener and SystemPowerStateListener comments to not rely on specific Power State definitions. • Change DevicePowerState and SystemPowerState interface comments to explain that these definitions are not mandatory but should be taken as examples.
0.3	Jan 21 2005	Add PowerPermission class and additional comments/explanations along the document

1.0	Nov 8 2005	Refine the API proposal
1.1	Nov 18 2005	Proposal updated to incorporate comments from Boeblingen meeting: • Add device capabilities • Add device mapping • Change sequence diagrams • Add device implementation recommendation • Improve definition of System and Device events • Update and Improve JavaDoc
1.2	March 1, 2006	Add 5.6 section to describe system power state used in the vehicle

1 Introduction

This document defines a technical solution to the requirements listed in RFP 41 – Power management [4].

The chapters 2, 3 and 4 are copied from the RFP.

This technical solution is based on the definition of two different groups of interface: one group handles the system power state, the second one handles device power states. This document defines as well the main entity: Power Manager that has the task to maintain a consistency between the system power state and the different device power states.

2 Application Domain

The parties, which are affected by the power management, include software components, such as: Operating System (OS), native applications, OSGi Framework, bundles, and services provided by these bundles, but also hardware components, such as system boards (e.g. processor) and embedded devices (i.e. GSM device). The software components adjust their processes according to the current power states, but they may also control hardware components and set them into different operation states (i.e. ON, OFF, SLOW).

Such hardware and software is used in:

- The automotive industry for off-board navigation, tracking, tracing and fleet management;
- Mobile device networks for delivery of value-added services to network users.

Software components (i.e. bundles) may change related hardware components operation states based on different power levels (i.e. BatteryLow). However, it is crucial to note that this type of power policy management should not be considered as power management in terms of this document. Here power management refers to management of OSGi-related mechanisms and entities such as:

1. Manages the state of the power source (sense and control of device and system power states)
2. Broadcasts power state changes to all interested power-aware entities

2.1 Standard references

Some of the features present in existing power management specifications can be used.

2.1.1 JavaPhone API

The Power Management package from the JavaPhone Specification (javax.power.management) [7] allows an application to monitor and respond to changes of power states in the system. Applications:

- can respond to various conditions such as full power, power managed for efficiency, suspend, and sleep states;

- are able to retrieve the current battery level, an estimate of remaining battery life, and whether an external power source is being used;

- are notified when the battery power is running low and a service is about to be terminated as a result.

2.1.2 AMI-C

AMI-C is a worldwide organization of motor vehicle manufacturers created to facilitate the development, promotion and standardization of automotive multimedia interfaces to motor vehicle communication networks.

AMI-C physical specification [8] defines three power modes: ON, OFF and SLEEP. ON supports all component or device functions on the network. OFF implies that all component or device functions have been shut down. SLEEP results in low power consumption from components or devices, with no data communications occurring. It also describes the transitions between modes through signal changing, the expected behavior from devices and the utilized management mechanisms for low consumption.

2.1.3 ACPI

Advanced Configuration and Power Interface (ACPI) [5] which was developed by Intel, Toshiba, and Microsoft, enables operating systems to “intelligently” manage power consumption by turning on devices only when needed, calculating remaining battery life, and controlling the speed of the processor depending on application requirements.

ACPI has four global system states - Mechanical Off, Soft Off, Sleeping, Working; five sleeping states; different device and CPU power and performance states; platform configurations; device capabilities; thermal and battery management.

3 Problem Description

Some systems have a mechanism for reacting on power events, when the battery is critical. According to power states of one or more power sources, OSGi components need to be notified. The monitoring of power states is outside of scope of this document. A PDA is a typical example. The PDA user needs to be notified according the power state of the battery, so that he has the chance to react properly, e.g. he can charge the battery with an external power supply, in order to prevent a shut down. It is crucial that the OSGi framework and bundles take care on their own. The problem is what action should be taken to shutdown the framework and all the bundles safely.

Depending on the device’s function, it may need to shut down at a later time, perhaps after it finishes a task, after some programmed time period, or after appropriate shutdown chores are completed.

In some cases the system and the controlled devices consume power from the same source. To prevent quick shutdown of the whole system, it is preferable to turn off only some of the controlled devices instead of all. Functions that require some input may be switched off after a predefined period of inactivity.

Additionally, there may be several power lines in the system. Each power line can have a set of devices attached to it, managed by a set of bundles, which in terms of power policy may all be treated the same way, or may have separate policies. The most appropriate example may be a car with multiple CAN networks, states of which are changed for each step of the ignition key. In the final stage there may also be other networks, like MOST, which will be initialized, after the most crucial ones have been started. Each network may go through different startup phases and power states, but they also may have been disabled. In this case it would be redundant to start all the related bundles, but rather we should start them when the network is powered up. This function is part of power policy management and outside the scope of OSGi power management.

3.1 Power Management

Receiving events for different power state changes is considered to be the core of the problem. Based on these events interested parties may decide to store their vital data, request a safe state of the managed device or simply notify the user.

Some parties may also invoke the economic power mode of their devices, according to the power states of one or more power sources. For example, a PDA may choose to lower the brightness of the LCD display or even turn it off, based on the user-defined power levels.

As stated previously, power management stands for management of OSGi-related mechanisms and entities, and not (explicitly) for power policy management. It receives power state events and is taking appropriate actions to inform all power-aware applications about the power-state change.

3.2 Power Management Entities

We may define several Power Management entities:

3.2.1 Power Manager

It controls the overall set of power states (system and allowed device power states). Triggered by a native power management component which will send trigger events important for Java app´s (e.g. system sleep, wakeup, device can run own power policy etc.).

3.2.2 Power State

We may define two kind of power states related to the type of system that they reference: System Power States are related to the whole system (OSGi Gateway). Devices Power States are related to the devices that has their own power management (e.g. car radio can allow running 1 hour after shut down car engine).

3.2.2.1 System Power States

System Power States are based on Global System States and Sleeping States from ACPI:

S0/G0: Working
S1/G1: Sleeping State
S2: Sleeping State
S3: Sleeping State
S4: Sleeping State
S5/G2: Soft Off

See complete definition in 5.4.1

3.2.2.2 Device Power States

Device Power States are defined using ACPI definition:

D0: Device completely active;
D1: Device is power managed, D1 state is device specific, Consume less power than D1;

D2: Device is power managed, D1 state is device specific, Consume less power than D2;
D3: Device is fully off.

See complete definition in 5.4.2

3.3 Use cases

3.3.1 Power Related Events (Triggers)

Use Case Name	**System Wakeup/Shutdown Based on Trigger Events**

Feature					
Actors	*In car user*	*External event*			
	X	*X*			
Description	The intent of the use case is to receive information about events related to car and Wakeup/Shutdown the system based on these events.				
Flow of events	**Basic flow – Wakeup the system** 1. The in car user (or sensor) make change in the car state – for example: • Ignition key on • Remote control system on • Bus request • e-call / b-call button • device power button on • media insertion 2. The Power Manager detects changes and generate trigger event related to the change. 3. Based on this event the system goes from suspended (or power off) to the running mode. 4. The use case ends.				
Alternate Flow	**Alternate flow – Shutdown the system** 1. The in car user (or sensor) make change in the car state – for example: • Lock car • Key off • Remote off 2. The Power Manager detects changes and generate trigger event related to the change. 3. Based on this event the system goes from running to the suspended (or power off) mode. 4. The use case ends.				

Must have devices	*N/A*
May have devices	Devices connected to the system.
Prior conditions	Service is running.
Post conditions	**Basic flow:** 1. System is in running mode **Alternative flows:** 1. System is in suspended (power off) mode
Notes	
Future considerations	Define reacting of receiving of some king of event
Lower level Use Cases	*N/A*
Date	29/06/2004
Status	Draft

3.3.2 Receive Power Source Event

Use Case Name	Generate Power Source Event

Feature					
Actors	*In car systems*				
	X				
Description	The intent of the use case is to receive information about events related to the changes of power source consumption – for example level of the battery.				
Flow of events	**Basic flow – generate Power Source Event** 1. The event related to the changes in three types of power sources should be generated: • Current OSGi gateway power source; • External power source; • Another OSGi gateway power source. 2. Events should bear enough information regarding the corresponding power source and one or more values. 3. The use case ends.				
Must have devices	*N/A*				
May have devices	Devices connected to the system.				
Prior conditions	Service is running.				
Post conditions	**Basic flow:** 1. Power Source Event is generated.				
Notes					
Future considerations	*N/A*				
Lower level Use Cases	*N/A*				
Date	29/06/2004				
Status	Draft				

3.3.3 Retrieve Power Source State

Use Case Name	Retrieve Power Source State

Feature					
Actors	*In car user*				
	X				
Description	The intent of the use case is to receive information about current status of the power sources.				
Flow of events	**Basic flow – retrieve Power Source state** 1. User wants to know Power Source (e.g Battery). 2. User requests receiving of the power source state. 3. Power Manager service retrieves information about state of the Power source. 4. Power source state is displayed to the user 5. The use case ends.				
Must have devices	*N/A*				
May have devices	Devices connected to the system.				
Prior conditions	Service is running.				
Post conditions	**Basic flow:** 1. Power Source state is returned.				
Notes					
Future considerations	*N/A*				
Lower level Use Cases	*N/A*				
Date	29/06/2004				
Status	Draft				

4 Requirements

1. For each device connected to the system or gateway itself we must be able to receive related device power state changes.
2. We must be able to retrieve the current system and device power state.
3. A common set of system power state and device power state should be defined. The definition should be extendible and should use existing standards.
4. An external application should be able to change the system power state, given that they have the permissions to do so.
5. An external application should be able to change a device power state given that they have the permissions to do so.

5 Technical Solution

5.1 Introduction

The Power Management specification defines the way the system and device power states are handled within the OSGi Service Platform.

There are clearly two different kinds of power state:

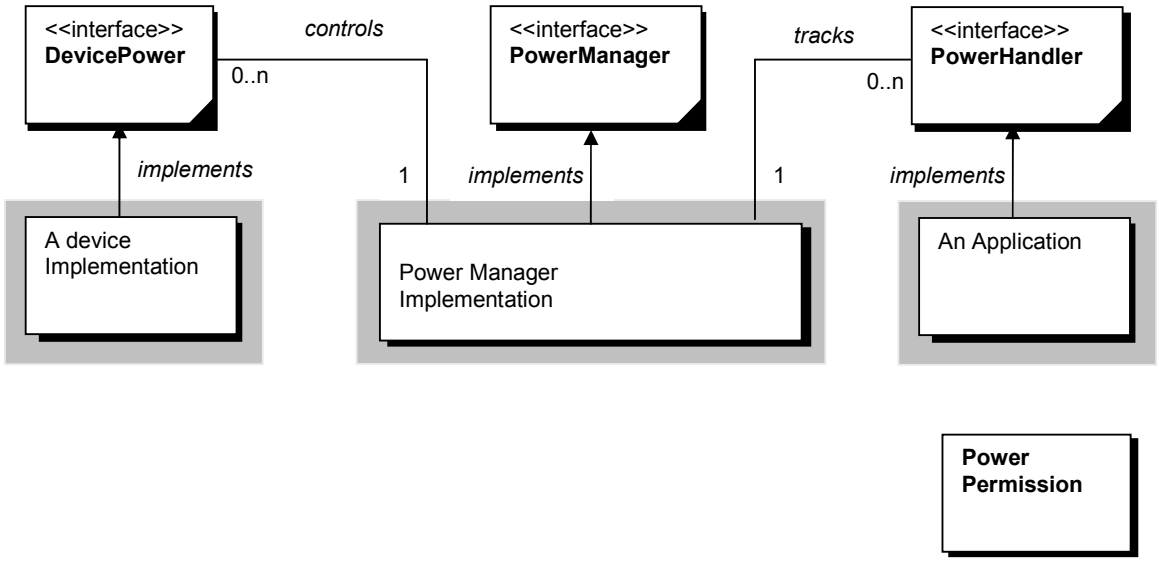
- System Power state (see definition below)
- Device Power state (see definition below)

The goals of this specification is to define the set of power states that a developer has to take into account and how these states are broadcasted within the OSGi Service Platform.

5.2 Entities

- *DevicePower* – A service that manages the power state of a specific device.
- *Device Power State* – The list of possible power state for a device. This list is extensible but the first set is based on the ACPI specification. The standard device power states are defined in the `DevicePower` interface by the following constants: D0, D1, D2 and D3.
- *PowerManager* – A service that a vendor has to implement to manage the overall power state changes. The Power Manager has to maintain a consistency between the current power state of the host (known as system power state) and power state of devices (known as device power state) connected to it.
- *PowerHandler* – A service that have to be registered within the framework to be part of query sequence when the `PowerManager` is moving to another power state.
- *System Power State* – The list of possible power state for the system. This list is extensible but the first set is based on the ACPI specification. The standard values are defined in the `PowerManager` interface by the following values: S0 (G0), S1, S2, S3, S4 and S5 (G2).

5.3 Class Diagram



5.4 System and Device Power States

The System Power States are defined by int values in `PowerManager` interface and Device Power States are defined by int values in `DevicePower` interface.

Nevertheless, the int values defined in these interfaces are reserved; they can not have another meaning.

Important Note

The sections 5.4.1 and 5.4.2 are a copy of ACPI specification.

5.4.1 System Power States

In the ACPI specification the System Power States are defined by Global System States G0 to G3 and Sleeping States S0 to S5

S0/G0 Working

A computer state where the system dispatches user mode (application) threads and they execute. In this state, peripheral devices (peripherals) are having their power state changed dynamically. The user can select, through some UI, various performance/power characteristics of the system to have the software optimize for performance or battery life. The system responds to external events in real time. It is not safe to disassemble the machine in this state.

S1/G1 Sleeping State

The S1 sleeping state is a low wake latency sleeping state. In this state, no system context is lost (CPU or chip set) and hardware maintains all system context.

S2 Sleeping State

The S2 sleeping state is a low wake latency sleeping state. This state is similar to the S1 sleeping state except that the CPU and system cache context is lost (the OS is responsible for maintaining the caches and CPU context). Control starts from the processor's reset vector after the wake event.

S3 Sleeping State

The S3 sleeping state is a low wake latency sleeping state where all system context is lost except system memory. CPU, cache, and chip set context are lost in this state. Hardware maintains memory context and restores some CPU and L2 configuration context. Control starts from the processor's reset vector after the wake event.

S4 Sleeping State

The S4 sleeping state is the lowest power, longest wake latency sleeping state supported by ACPI. In order to reduce power to a minimum, it is assumed that the hardware platform has powered off all devices. Platform context is maintained.

S5/G2 Soft Off State

The S5 state is similar to the S4 state except that the OS does not save any context. The system is in the "soft" off state and requires a complete boot when it wakes. Software uses a different state value to distinguish between the S5 state and the S4 state to allow for initial boot operations within the BIOS to distinguish whether or not the boot is going to wake from a saved memory image.

From ACPI definition we use the following System Power States:

- **S0**/G0 (WORKING)
- **S1**/G1 (SLEEPING)
- **S2** (SLEEPING)
- **S3** (SLEEPING)
- **S4** (SLEEPING)
- **S5**/G2 (SOFT_OFF)

Note that the MECHANICAL_OFF (G3) Global System State from ACPI is not used in this API because it is not relevant for applications.

5.4.2 Device Power States

D3 Off

Power has been fully removed from the device. The device context is lost when this state is entered, so the OS software will reinitialize the device when powering it back on. Since device context and power are lost, devices in this state do not decode their address lines. Devices in this state have the longest restore times. All classes of devices define this state.

D2

The meaning of the D2 Device State is defined by each device class. Many device classes may not define D2. In general, D2 is expected to save more power and preserve less device context than D1 or D0. Buses in D2 may cause the device to lose some context (for example, by reducing power on the bus, thus forcing the device to turn off some of its functions).

D1

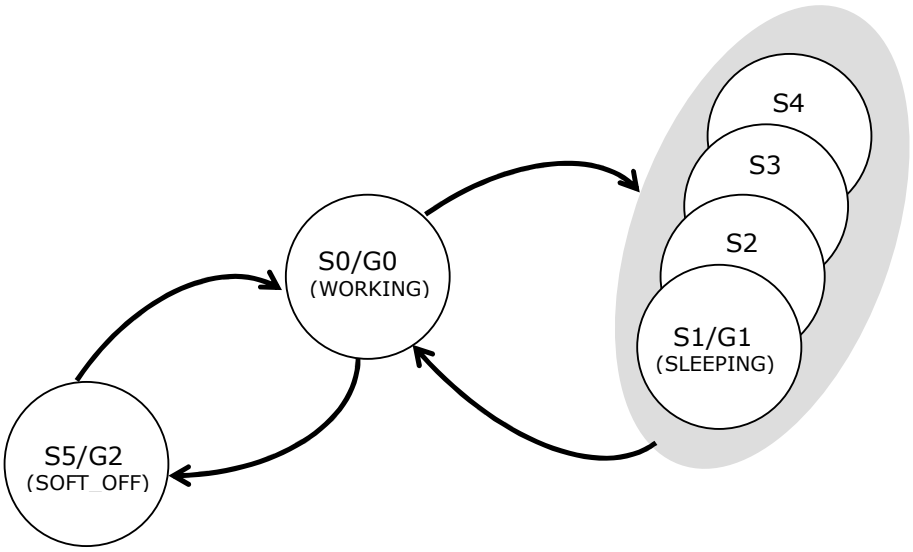
The meaning of the D1 Device State is defined by each device class. Many device classes may not define D1. In general, D1 is expected to save less power and preserve more device context than D2.

D0 Fully-On

This state is assumed to be the highest level of power consumption. The device is completely active and responsive, and is expected to remember all relevant context continuously.

5.5 System Power State Transitions

5.5.1 Transition diagram



5.5.2 Transition descriptions

S5 (SOFT_OFF) -> S0 (WORKING)

The system is starting up. Applications will be informed of the working state at the end of the transition.

S0 (WORKING) -> S5 (SOFT_OFF)

The system is shutting down. Applications will be queried to accept of reject the transition. In case of a successful query an event is sent before the transition starting the transition.

S0 (WORKING) -> Sx (SLEEPING States)

Before moving to a Sleeping State (S1-S4), the applications are queried to accept or reject the transition. In case of a successful query an event is sent before starting the transition.

Sx (SLEEPING States) -> S0 (WORKING)

According to a wake event the system is resumed to the working state. Applications are informed of the working state at the end of the transition.

5.5.3 Power Event Topics

The Power Manager and Device Power services use the Event Admin service to send information about power changes.

SystemEvent topic	Comments
`org/osgi/service/power/SystemEvent/S0`	Indicates that the system is in S0 (WORKING) power state.
`org/osgi/service/power/SystemEvent/TO_S1`	Indicates that system is currently moving to S1 (SLEEPING_S1) power state.
`org/osgi/service/power/SystemEvent/TO_S2`	Indicates that system is currently moving to S2 (SLEEPING_S2) power state.
`org/osgi/service/power/SystemEvent/TO_S3`	Indicates that system is currently moving to S3 (SLEEPING_S3) power state.
`org/osgi/service/power/SystemEvent/TO_S4`	Indicates that system is currently moving to S4 (SLEEPING_S4) power state.
`org/osgi/service/power/SystemEvent/TO_S5`	Indicates that system is currently moving to S5 (SOFT_OFF) power state.

Additional properties are defined for SystemEvent:

- `bundle` The source bundle.
- `service` The source's ServiceReference
- `timestamp` The timestamp.

DeviceEvent topic	Comments
`org/osgi/service/power/DeviceEvent/D0`	Indicates that the device has changed its power state to D0
`org/osgi/service/power/DeviceEvent/D1`	Indicates that the device has changed its power state to D1
`org/osgi/service/power/DeviceEvent/D2`	Indicates that the device has changed its power state to D2
`org/osgi/service/power/DeviceEvent/D3`	Indicates that the device has changed its power state to D3

An additional property is needed for device power events:

- `bundle` The source bundle.
- `service` The source's ServiceReference
- `service.pid` The source's DevicePower PID
- `timestamp` The timestamp.

5.6 System Power State description in the vehicle

(S0) - The vehicle system is up and running. All devices are powered on.
(S1) - The vehicle system is in sleep mode and should wake up quickly. Some devices can be powered for specific needs.
(S2) - Usually not handled
(S3) - Usually not handled

(S4) - The vehicle system is in a hibernate state. It needs time to wake up because contexts must be restored and all applications have to be resynchronized.
All devices are powered off.
(S5) - The vehicle system is off.

5.7 Power Management scenarios

The following paragraph describes several scenarios that are relevant for Power Management.

5.7.1 Startup the system (S5 -> S0)

The system is off and the power button is pressed. The framework is restarted, as soon as the framework sent an event `FrameworkEvent.STARTED`, the Power Manager sends an event (`org/osgi/service/power/SystemEvent/S0`) to inform all event handlers that the transition is complete and the system is up and running.

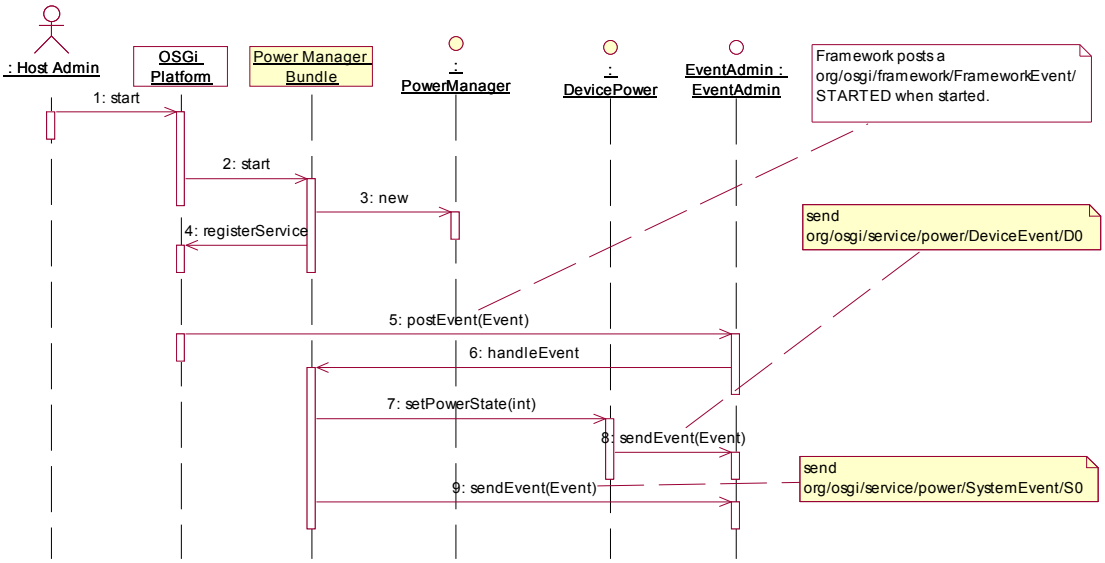


Figure 1: Startup the system

5.7.2 Shutdown the system (S0-> S5)

The system will shutdown, a client bundle asks the Power Manager to set the power state to the S5 state. The Power Manager queries sequentially the transition to all Power Handlers. When all handlers have positively responded to this query, the Power Manager sends an event (`org/osgi/service/power/SystemEvent/TO_S5`) to inform all Event Handlers that the transition has been started. The transition cannot be canceled or stopped. The return value of setPowerState() is null. The framework is stopped.

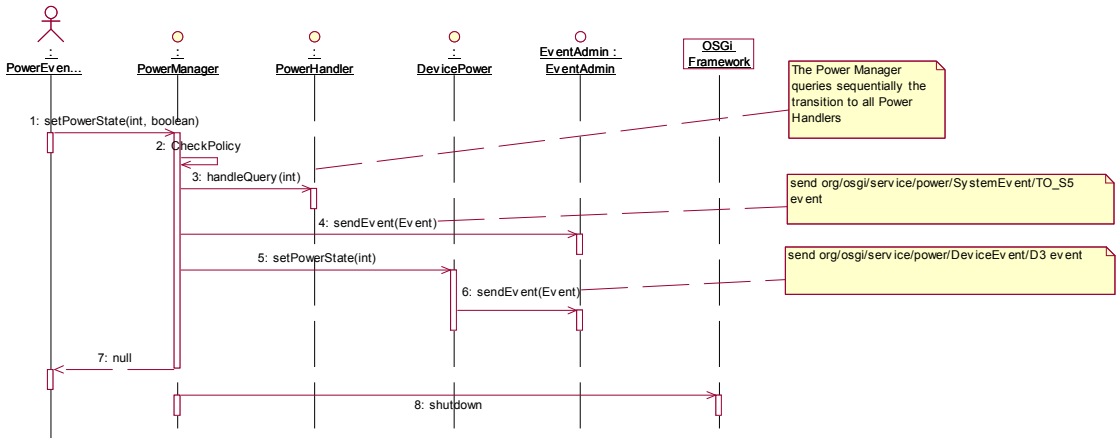


Figure 2: Shutdown the system

5.7.3 Shutdown the system with veto (S0->S5)

The system will shutdown, a client bundle asks the Power Manager to set the power state to the S5 state. The Power Manager queries sequentially the transition to all Power Handlers. If at least one of the handlers denies the query, the Power Manager informs in reverse order all handlers previously called to inform them that the query failed. The return value of `setPowerState() is` an array of ServiceReference which have denied the request.

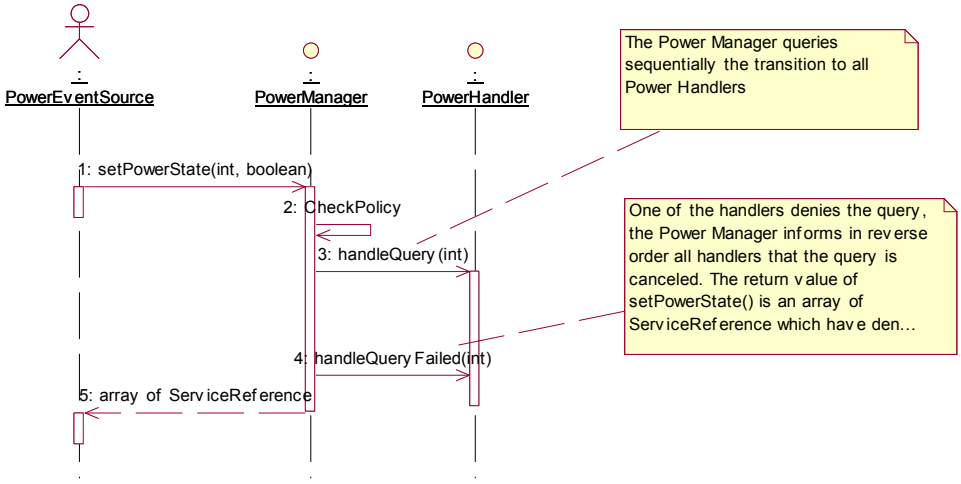


Figure 3: Shutdown the system with veto

5.7.4 Suspend the system (S0->Sx)

A client bundle is asking the Power Manager to set the Power State to Sx (x being from 1 to 4) state. The Power Manager queries sequentially the transition to all Power Handlers. When all handlers have positively answered to this query, the Power Manager sends an event (`org/osgi/service/power/SystemEvent/TO_SX`) to inform all event handlers that a transition has been started. The transition cannot be canceled or stopped. The return value of `setPowerState()` is null.

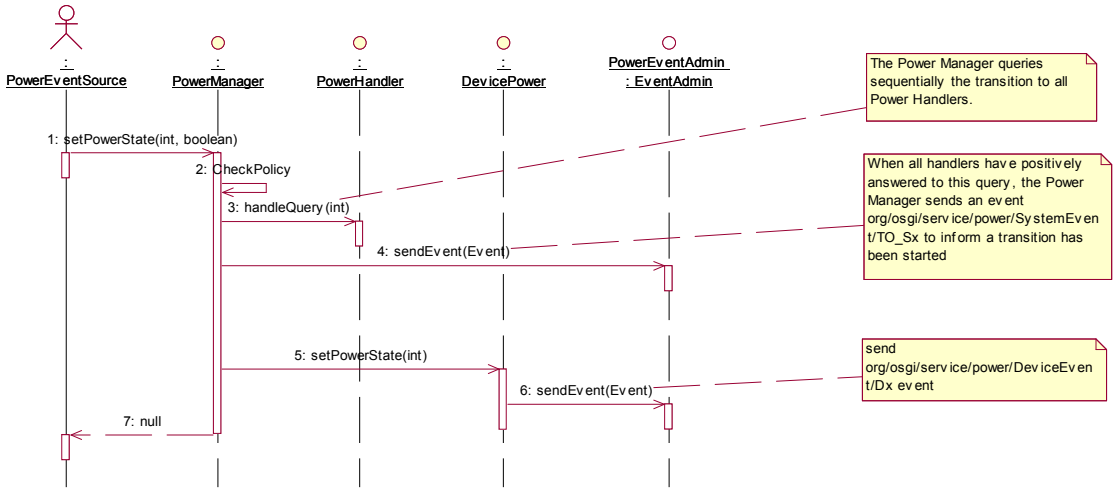


Figure 4: Suspend the system

5.7.5 Suspend the system with veto (S0->Sx)

The system will suspend, a client bundle asks the Power Manager to set the power state to Sx (x being from 1 to 4) state. The Power Manager queries sequentially the transition to all Power Handlers. if at least one of the handlers denies the query, the Power Manager informs in reverse order all handlers previously called to inform them that the query failed. The return value of `setPowerState()` is an array of ServiceReference which have denied the request.

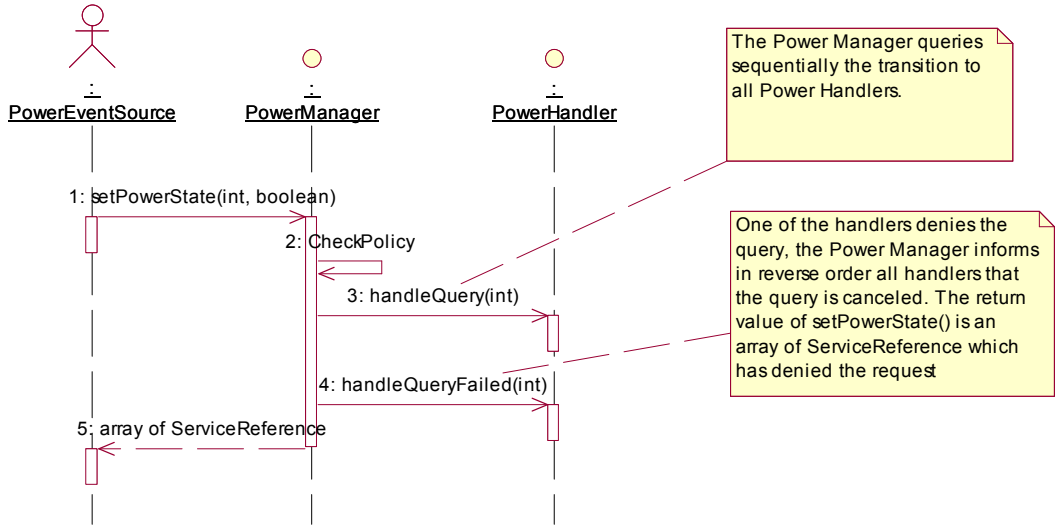


Figure 5: Suspend the system with veto

5.7.6 Resume the system (Sx->S0)

A client bundle is asking the Power Manager to set the Power State to S0 state. The Power Manager sends an event (`org/osgi/service/power/SystemEvent/S0`) to inform all event handlers that a transition has been started. The transition cannot be canceled or stopped. The return value of `setPowerState()` is null.

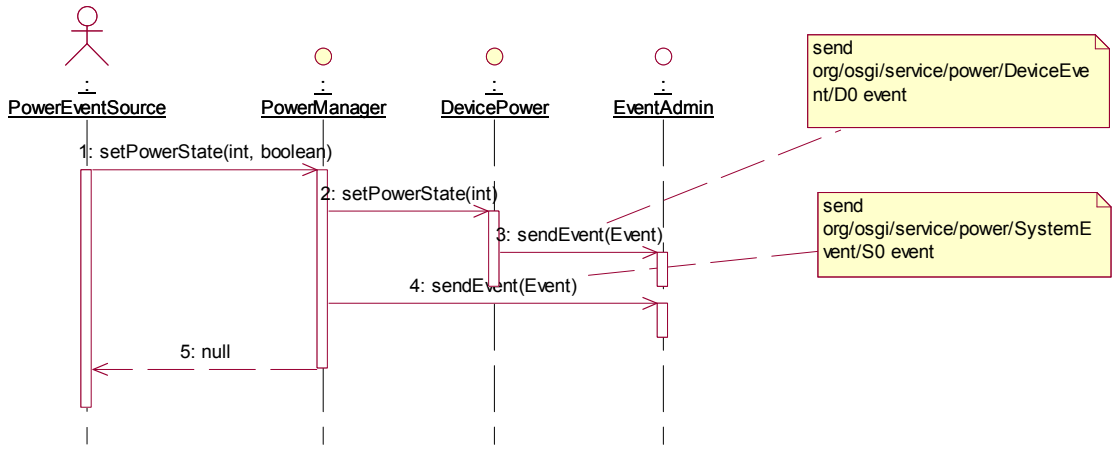


Figure 6: Resume the system

5.8 Power Manager Implementation Recommendations

The implementation of Power Manager is completely vendor dependent.

The policy to manage:

- the Power Manager event handler order;
- the consistency between System and Device Power States;

is free.

However we recommend the following default mapping between System and Device Power States:

System Power States	Device Power State
S0 (WORKING)	D0
S1 (SLEEPING)	D3
S2 (SLEEPING)	D3
S3 (SLEEPING)	D3
S4 (SLEEPING)	D3
S5 (SOFT_OFF)	D3

Power Manager will use this default mapping when the device does not define it.

The Power Manager will not process several requests (`setPowerState`) in parallel. When a `setPowerState` is called it has to be finished before processing another request.

5.9 Device Power Implementation

A device that has power capabilities has to implement `DevicePower` interface and register the service within the framework with the following optional service properties registration:

- power.device.capabilities: A service property registration for `DevicePower`. It indicates the list of device power state supported, one or several of the following values: D0, D1, D2 or D3. The type of this property must be an Integer. It represents a mask of all device supported states. If this property is not used then it assumes that the `DevicePower` only supports D0 and D3 (It corresponds to a mask value set to 9 (mask = Integer(`DevicePower.D0` | `DevicePower.D3`)).

- power.device.mapping: A service property registration for `DevicePower`. It indicates the mapping (System/Device Power States) that the device wants to propose to the Power Manager. The Power Manager is allowed to override it. The type of the property must be int[]. Each index position represents a system power state from S0 to S5. Each value of the array represents the `DevicePower` state value for a given system power state position. For example, the following mapping

  S0 -> D0
  S1 -> D1
  S2 -> unspecified
  S3 -> D3
  S4 -> D3
  S5 -> D3

  is represented as follows: int[] mappings = new int[]{`DevicePower.D0`, `DevicePower.D1`, `DevicePower.UNSPECIFIED`, `DevicePower.D3`, `DevicePower.D3`, `DevicePower.D3` }

  If this property is not used or invalid then the default mapping (defined in Power Manager) is used.

5.9.1 Example

As an example, a in-vehicle system has a TMC Tuner that supports the following states:

• D0 – Full, tuner is up and running

• D1 – Tuner records Traffic messages for a configurable time

• D3 – Tuner is off.

The power manager can control the timing of the states. Its policy can require that the TMC tuner has to be able to record traffic messages for a certain time.

The TMC Tuner driver must therefore implement a Device Power service. The following code shows how this could be done.

```
public class TMCTunerDriver implements BundleActivator, DevicePower {
           /**
            * Current device power state
            */
           private int
           powerstate;

           /**
            * Registration object for our DevicePower
            */
           private ServiceRegistration     reg;
```

```
            /**
             * Bundle context reference
             */
            private BundleContext           context;

            /**
             * Called when bundle is starting
             *
             * @param context bundle context
             */
            public void start(BundleContext context) {
                        // initialize
                        int[] map = new int[6];
                        map[PowerManager.S0] = D0;
                        map[PowerManager.S1] = D1;
                        map[PowerManager.S2] = D1;
                        map[PowerManager.S3] = D1;
                        map[PowerManager.S4] = D1;
                        map[PowerManager.S5] = D1;
                        Hashtable ps = new Hashtable();
                        ps.put(Constants.SERVICE_PID, getPID());
                        ps.put("power.device.capabilities", new Integer(D0 | D1 |
D3));
                        ps.put("power.device.mapping", map);
                        reg = context.registerService(DevicePower.class.getName(),
this, ps);
            }

            /**
             * Called when bundle is stopping
             *
             * @param context bundle context
             */
            public void stop(BundleContext context) {
            }

            /**
             * Returns the current power state for this device
             *
             * @return the current power state for this device
             */
            public int getPowerState() {
                        return powerstate;
            }

            /**
             * Changes the device power state with the given value
             *
             * @param state power state value
             */
            public void setPowerState(int state) {
                        String stateName = getStateName(state);
                        switch (state) {
                                    case D0 :
                                                stopRecordTMCInfo();
                                                break;
                                    case D1 :
                                                startRecordTMCInfo();
                                                break;
                                    case D2 :
```

```
                                        throw new
IllegalArgumentException("Power state value not allowed "
                                                                + state);
                                case D3 :
                                        stopRecordTMCInfo();
                                        break;
                                default :
                                        throw new
IllegalArgumentException("Illegal power state value "
                                                                + state);
                        }

                        Dictionary map = new Hashtable();
                        map.put("bundle", context.getBundle());
                        map.put("service", reg.getReference());
                        map.put("service.pid", getPID());
                        map.put("timestamp", new Long(System.currentTimeMillis()));
                        Event event = new
Event("org/osgi/service/power/DeviceEvent/"
                                        + stateName, map);
                        EventAdmin admin = getEventAdmin();
                        admin.sendEvent(event);
                }

                /**
                 * Returns the string representation of the given Power state.
                 *
                 * @return the string representation of the given Power state.
                 */
                private String getStateName(int state) {
                        String name = null;
                        switch (state) {
                                case D0 :
                                        name = "D0";
                                        break;
                                case D1 :
                                        name = "D1";
                                        break;
                                case D2 :
                                        name = "D2";
                                        break;
                                case D3 :
                                        name = "D3";
                                        break;
                                default :
                                        name = "??";

                        }
                        return name;
                }

                /**
                 * Returns the PID for this device power
                 *
                 * @return the PID for this device power
                 */
                private String getPID() {
                        return "com.acme.tmctuner.driver";
                }
```

```
        /**
         * Returns the Event Admin service object
         *
         * @return the Event Admin service object
         */
        private EventAdmin getEventAdmin() {
                /* ... */
                return null;
        }

        /**
         * Starts recording TMC infos
         */
        private void startRecordTMCInfo() {
                /* ... */
        }

        /**
         * Stops recording TMC infos
         */
        private void stopRecordTMCInfo() {
                /* ... */
        }
}
```

5.10 Java API

5.10.1 Package org.osgi.service.power

The OSGi Power Manager Package.

See:

Description

Interface Summary	
DevicePower	Interface for identifying Power Device as defined in ACPI Specifications.
PowerHandler	A service that has to be registered within the framework to be part of query sequence when the `PowerManager` is moving to another system power state.
PowerManager	An interface that a vendor has to implement to manage the overall power state changes.

Class Summary	
PowerPermission	Permission to set the system or/and device power states managed repectively by `PowerManager` and `DevicePower` services.

5.10.2 Package org.osgi.service.power Description

The OSGi Power Manager Package. Specification Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. For example:

```
Import-Package: org.osgi.service.power; version=1.0
```

5.10.3 org.osgi.service.power
Interface DevicePower

public interface **DevicePower**

Interface for identifying Power Device as defined in ACPI Specifications.

A service must implement this interface to indicate that it is a device power service.

Field Summary

static int	**D0** 'D0' Device Power state as defined in ACPI Specifications.
static int	**D1** 'D1' Device Power State as defined in ACPI Specifications.
static int	**D2** 'D2' Device Power State as defined in ACPI Specifications.
static int	**D3** 'D3' Device Power State as defined in ACPI Specifications.
static java.lang.String	**DEVICE_POWER_CAPABILITIES** Service registration property key (named `power.device.capabilities`) for DevicePower indicating the list of supported device power states, one or several of the following values: `D0`, `D1`, `D2` or `D3`.
static java.lang.String	**DEVICE_POWER_MAPPING** Service registration property key (named `power.device.mapping` for DevicePower indicating the mapping (System/Device Power States) that the device wants to propose to the Power Manager.
static int	**UNSPECIFIED_STATE** Unspecified device power state.

Method Summary

int	**getPowerState**() Returns the current power state of this device power.
void	**setPowerState**(int state) Sets the device power state with the given value.

Field Detail

D0

```
public static final int D0
```

'D0' Device Power state as defined in ACPI Specifications.

This state is assumed to be the highest level of power consumption. The device is completely active and responsive, and is expected to remember all relevant context continuously.

See Also:
Constant Field Values

D1

```
public static final int D1
```

'D1' Device Power State as defined in ACPI Specifications.

The meaning of the D1 Device State is defined by each device class. Many device classes may not define D1. In general, D1 is expected to save less power and preserve more device context than D2.

See Also:
Constant Field Values

D2

```
public static final int D2
```

'D2' Device Power State as defined in ACPI Specifications.

The meaning of the D2 Device State is defined by each device class. Many device classes may not define D2. In general, D2 is expected to save more power and preserve less device context than D1 or D0. Buses in D2 may cause the device to lose some context (for example, by reducing power on the bus, thus forcing the device to turn off some of its functions).

See Also:
Constant Field Values

D3

```
public static final int D3
```

'D3' Device Power State as defined in ACPI Specifications.

Power has been fully removed from the device. The device context is lost when this state is entered, so the OS software will reinitialize the device when powering it back on. Since device context and power are lost, devices in this state do not decode their address lines. Devices in this state have the longest restore times. All classes of devices define this state.

See Also:
Constant Field Values

UNSPECIFIED_STATE

```
public static final int UNSPECIFIED_STATE
```

Unspecified device power state. This state can be used in the `DEVICE_POWER_MAPPINGS` property registration to express that the transition is not defined.

See Also:
Constant Field Values

DEVICE_POWER_CAPABILITIES

`public static final java.lang.String` **`DEVICE_POWER_CAPABILITIES`**

Service registration property key (named `power.device.capabilities`) for DevicePower indicating the list of supported device power states, one or several of the following values: `D0`, `D1`, `D2` or `D3`. The type of this property must be an `Integer`. It represents a mask of all supported device power states. If this property is not used then `PowerManager` assumes that this DevicePower only supports `D0` and `D3` states.

See Also:

Constant Field Values

DEVICE_POWER_MAPPING

`public static final java.lang.String` **`DEVICE_POWER_MAPPING`**

Service registration property key (named `power.device.mapping` for DevicePower indicating the mapping (System/Device Power States) that the device wants to propose to the Power Manager. The Power Manager is allowed to override it. The type of the property must be int[]. Each index position represents a system power state from S0 to S5. Each value of the array represents the DevicePower state value for a given system power state position. For example, the following mapping

```
S0 -> D0
S1 -> D1
S2 -> unspecified
S3 -> D3
S4 -> D3
S5 -> D3
```

is represented as follows: `int[] mappings = new int[]{D0, D1, UNSPECIFIED STATE, D3, D3, D3};` If this property is not used or invalid then the default mapping (defined in Power Manager) is used.

See Also:

Constant Field Values

Method Detail

getPowerState

`public int` **`getPowerState`**`()`

Returns the current power state of this device power.

Returns:

the current device power state.

setPowerState

```
public void setPowerState(int state)
                  throws java.lang.SecurityException,
                         java.lang.IllegalArgumentException
```

Sets the device power state with the given value. This method is generally called by `PowerManager` for a transition to another system power state.

Parameters:

`state` - the device power state that the device must transit to.

Throws:

`java.lang.IllegalArgumentException` - if the given state value is not one of the device power states or the transition is not allowed.

`java.lang.SecurityException` - If the caller does not have the appropriate `PowerPermission[this,setDevicePower]`, and the Java Runtime Environment supports permissions.

See Also:
PowerPermission

5.10.4 org.osgi.service.power
Interface PowerHandler

public interface **PowerHandler**

A service that has to be registered within the framework to be part of query sequence when the `PowerManager` is moving to another system power state.

Version:
$Revision$

Method Summary

`boolean`	**`handleQuery`**`(int state)` Calls by the `PowerManager` to request permission to system power state change to the given value.
`void`	**`handleQueryFailed`**`(int state)` Called by the `PowerManager` to notify that the previous request change has been denied.

Method Detail

handleQuery

`public boolean` **`handleQuery`**`(int state)`

Calls by the `PowerManager` to request permission to system power state change to the given value.

Parameters:
`state` - requested state.

Returns:
`true` if the change is accepted, `false` otherwize.

handleQueryFailed

`public void` **`handleQueryFailed`**`(int state)`

Called by the `PowerManager` to notify that the previous request change has been denied. Only Power Handlers previouly notified on `handleQuery(int)` must be called on this method.

Parameters:
`state` - rejected state.

5.10.5 org.osgi.service.power
Interface PowerManager

public interface **PowerManager**

An interface that a vendor has to implement to manage the overall power state changes. The Power Manager has to maintain a consistency between the current power state of the host (known as system power state) and power state of devices (known as device power state) connected to it.

There will only be a single instance of this service registered with the Framework.

Version:
$Revision$

Field Summary

`static int`	**`S0`** S0 (Working or G0) system power state as defined in ACPI Specifications.
`static int`	**`S1`** S1 Sleeping system power state as defined in ACPI Specifications.
`static int`	**`S2`** S2 Sleeping system power state as defined in ACPI Specifications.
`static int`	**`S3`** S3 Sleeping system power state as defined in ACPI Specifications.
`static int`	**`S4`** S4 Sleeping system power state as defined in ACPI Specifications.
`static int`	**`S5`** S5 system power state (Soft Off or G2) as defined in ACPI Specifications.

Method Summary

`int`	**`getPowerState`**`()` Returns the current system power state.
`org.osgi.framework.ServiceReference[]`	**`setPowerState`**`(int state, boolean urgent)` Changes the system power state with the given value.

Field Detail

S0

`public static final int` **`S0`**

S0 (Working or G0) system power state as defined in ACPI Specifications. A computer state where the system dispatches user mode (application) threads and they execute. In this state, peripheral devices (peripherals) are having their power state changed dynamically. The user can select, through some UI, various performance/power characteristics of the system to have the software optimize for performance or battery life. The system responds to external events in real time. It is not safe to disassemble the machine in this state.

See Also:
Constant Field Values

S1

`public static final int S1`

S1 Sleeping system power state as defined in ACPI Specifications. The S1 sleeping state is a low wake latency sleeping state. In this state, no system context is lost (CPU or chip set) and hardware maintains all system context.

See Also:
Constant Field Values

S2

`public static final int S2`

S2 Sleeping system power state as defined in ACPI Specifications. The S2 sleeping state is a low wake latency sleeping state. This state is similar to the S1 sleeping state except that the CPU and system cache context is lost (the OS is responsible for maintaining the caches and CPU context). Control starts from the processor's reset vector after the wake event.

See Also:
Constant Field Values

S3

`public static final int S3`

S3 Sleeping system power state as defined in ACPI Specifications. The S3 sleeping state is a low wake latency sleeping state where all system context is lost except system memory. CPU, cache, and chip set context are lost in this state. Hardware maintains memory context and restores some CPU and L2 configuration context. Control starts from the processor's reset vector after the wake event.

See Also:
Constant Field Values

S4

`public static final int S4`

S4 Sleeping system power state as defined in ACPI Specifications. The S4 sleeping state is the lowest power, longest wake latency sleeping state supported by ACPI. In order to reduce power to a minimum, it is assumed that the hardware platform has powered off all devices. Platform context is maintained.

See Also:
Constant Field Values

S5

`public static final int S5`

S5 system power state (Soft Off or G2) as defined in ACPI Specifications. The S5 state is similar to the S4 state except that the OS does not save any context. The system is in the "soft" off state and requires a complete boot when it wakes. Software uses a different state value to distinguish between the S5 state and the S4 state to allow for initial boot operations within the BIOS to distinguish whether or not the boot is going to wake from a saved memory image.

See Also:
Constant Field Values

Method Detail

getPowerState

`public int getPowerState()`

Returns the current system power state.
Returns:
the current system power state.

---

setPowerState

```
public org.osgi.framework.ServiceReference[] setPowerState(int state,
                                                           boolean urgent)
                                                    throws
java.lang.SecurityException,

java.lang.IllegalArgumentException
```

Changes the system power state with the given value.

The state must be one of the system power state values.

if `urgent` is set to `true`, all interested `PowerHandler` services are asked for request permission by calling `PowerHandler.handleQuery(int)` method. In case of veto from a `PowerHandler`, all previouly notified `PowerHandler`s must be called on `PowerHandler.handleQueryFailed(int)` to inform that the transition has been rejected. Then the method ends and `false` is returned. If not, the process continues. Then, Power Manager sents the appropriate event to `EventAdmin.sendEvent(Event)` for publishing the corresponding power state change.

Parameters:
`state` - the state into which the system is going to transit.
`urgent` - `true` to force the system to change state without requesting any permission from `PowerHandler`s. `true` to query all `PowerHandler`s for permission. In case of at least one of the `PowerHandler`s denies the query, the Power Manager informs in reverse order all `PowerHandler`s that the query failed and an array of `ServiceReference`s which rejected the request is returned. In circumstances like a low power or emergency shutdown, `urgent` should be set to `true`.
Returns:
array of `PowerHandler`s `ServiceReference`s which have denied the request; `null` otherwize.
Throws:
`java.lang.IllegalArgumentException` - if the given state value is not one of the system power states or the transition is not allowed.
`java.lang.SecurityException` - If the caller does not have the appropriate `PowerPermission[system,setSystemPower]`, and the Java Runtime Environment supports permissions.

5.10.6 org.osgi.service.power

Class PowerPermission

```
java.lang.Object
  └java.security.Permission
      └java.security.BasicPermission
          └org.osgi.service.power.PowerPermission
```

All Implemented Interfaces:
java.security.Guard, java.io.Serializable

---

public class **PowerPermission**
extends java.security.BasicPermission

Permission to set the system or/and device power states managed repectively by `PowerManager` and `DevicePower` services.

The permission name is the name (or name prefix). The naming convention follows the hierarchical naming convention. Also, an asterisk may appear at the end of the name, following a ".", or by itself, to signify a wildcard match. For example: "org.osgi.a.b.*" or "*" is valid, but "*a.b" or "a*b" are not valid. `PowerPermission` with the reserved name "system" represents the system power. `PowerPermission` with the reserved name "*" represents the permission required for setting any devices or system.

There are the following possible actions: `setSystemPower` and `setDevicePower`

- `setSystemPower` action allows a bundle to set the system power state.
- `setDevicePower` action allows a bundle to set the device power under that name.
- "*" can be used to set power states on devices and system.

Version:
$Revision: $
See Also:
Serialized Form

Field Summary

`static java.lang.String`	**`SET_DEVICE_POWER`** The action string `setDevicePower` (Value is "setDevicePower").
`static java.lang.String`	**`SET_SYSTEM_POWER`** The action string `setSystemPower` (Value is "setSystemPower").
`static java.lang.String`	**`SYSTEM`** The permission name "system" reprenting the system power.

Constructor Summary

`PowerPermission``(java.lang.String name, java.lang.String actions)` Create a new PowerPermission with the given name (may be wildcard) and actions.

Method Summary

`boolean`	**`equals`**`(java.lang.Object obj)` Determines the equalty of two PowerPermission objects.
`java.lang.String`	**`getActions`**`()` Returns the canonical string representation of the actions.
`int`	**`hashCode`**`()` Returns the hash code value for this object.
`boolean`	**`implies`**`(java.security.Permission p)` Determines if a `PowerPermission` object "implies" the specified permission.
`java.security.PermissionCollection`	**`newPermissionCollection`**`()` Returns a new `PermissionCollection` object for

	storing `PowerPermission` objects.
`java.lang.String`	**`toString`**`()` Returns a string describing this `PowerPermission`.

Methods inherited from class java.security.Permission
`checkGuard, getName`

Methods inherited from class java.lang.Object
`clone, finalize, getClass, notify, notifyAll, wait, wait, wait`

Field Detail

SYSTEM

```
public static final java.lang.String SYSTEM
```

The permission name "system" reprenting the system power.

See Also:

Constant Field Values

SET_SYSTEM_POWER

```
public static final java.lang.String SET_SYSTEM_POWER
```

The action string `setSystemPower` (Value is "setSystemPower").

See Also:

Constant Field Values

SET_DEVICE_POWER

```
public static final java.lang.String SET_DEVICE_POWER
```

The action string `setDevicePower` (Value is "setDevicePower").

See Also:

Constant Field Values

Constructor Detail

PowerPermission

```
public PowerPermission(java.lang.String name,
                       java.lang.String actions)
```

Create a new PowerPermission with the given name (may be wildcard) and actions.

Parameters:

`name` - identification of the system or the device. It can contain "*"

`actions` - `setSystemPower` or `setDevicePower`.

Method Detail

implies

```
public boolean implies(java.security.Permission p)
```

Determines if a `PowerPermission` object "implies" the specified permission.

Parameters:
`p` - The target permission to check.
Returns:
`true` if the specified permission is implied by this object; `false` otherwise.

---

getActions

`public java.lang.String` **`getActions`**`()`

Returns the canonical string representation of the actions. Always returns present actions in the following order: `setSystemPower`, `setDevicePower`.
Returns:
The canonical string representation of the actions.

---

newPermissionCollection

`public java.security.PermissionCollection` **`newPermissionCollection`**`()`

Returns a new `PermissionCollection` object for storing `PowerPermission` objects.
Returns:
A new `PermissionCollection` object suitable for storing `PowerPermission` objects.

---

equals

`public boolean` **`equals`**`(java.lang.Object obj)`

Determines the equalty of two PowerPermission objects. Checks that specified object has the same class name and action as this `PowerPermission`.
Parameters:
`obj` - The object to test for equality.
Returns:
true if obj is a `PowerPermission`, and has the same class name and actions as this `PowerPermission` object; `false` otherwise.

---

hashCode

`public int` **`hashCode`**`()`

Returns the hash code value for this object.
Returns:
Hash code value for this object.

---

toString

`public java.lang.String` **`toString`**`()`

Returns a string describing this `PowerPermission`. The convention is to specify the class name, the permission name, and the actions in the following format: '(org.osgi.service.power.PowerPermission "name" "actions")'.
Returns:
information about this `Permission` object.

6 Considered Alternatives

None

7 Security Considerations

There are two methods in the Power Management specifications where the appropriate permission must be checked. These two methods offer the possibility to request the change of the system power state or device power state.

```
ServiceReference[] PowerManager.setPowerState(int powerState, boolean urgent);
DevicePower.setPowerState(int powerState);
```

A `PowerPermission` class is used to allow a bundle to set a new power state.
This permission has to be created with a name and an action.

There are the following possible names:

- "`system`" (case insensitive) means that the bundle can perform a change (setPowerState) to the system.
- "*" means that the bundle can perform a change to the system and/or to any devices
- Any other string is considered as a service.pid and the bundle is allowed to change the power state to the device that has the same id.

There are the following possible actions:

- "`setSystemPower`" action allows a bundle to set the system power state.
- "`setDevicePower`" action allows a bundle to set the device power under that name.
- "*" can be used to set power states on devices and system.

8 Document Support

8.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. OSGi Service Platform - Core Specifications, Release 4.0, October 2005

[4]. RFP 41 – Power Management, http://membercvs.osgi.org/rfps/rfp-0041-Power_Management.doc.

[5]. ACPI Specifications - Revision 3.0, http://acpi.info

[6]. MNCRS Specification 1.1, http://membercvs.osgi.org/external/mncrs

[7]. JavaPhone 1.0 API specification, Sun Microsystems, Inc., http://java.sun.com/products/javaphone/

[8]. AMI-C 4001, AMI-C physical specification, v1.00, 2003-02-28

8.2 Author's Address

Name	Olivier Pavé
Company	Siemens VDO Automotive
Address	Bâtiment Alpha 80, route des lucioles – BP 305 06906 Sophia-Antipolis Cedex
Voice	+33 (0) 492 381 129
e-mail	olivier.pave@siemens.com

Name	Miguel Lopez
Company	Siemens VDO Automotive
Address	Bâtiment Alpha 80, route des lucioles – BP 305 06906 Sophia-Antipolis Cedex
Voice	+33 (0) 492 381 151
e-mail	miguel.lopez@siemens.com

8.3 Acronyms and Abbreviations

~~ACPI~~	~~Advanced Configuration & Power Interface~~

All Page Within in this Box

	ACPI is an open industry specification co-developed by Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. ACPI establishes industry-standard interfaces for OS-directed configuration and power management on laptops, desktops, and servers.
MNCRS	Mobile Network Computing Reference Specification

8.4 End of Document