Device Abstraction Layer

Draft

91 Pages

Abstract

Defines a new device abstraction API in OSGi platform. It provides a simple access to the devices and their functionality.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the "Distribution") in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback ("Feedback") on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2   Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3   Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4   Table of Contents

0.5   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 10.1.

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

0.6   Revision History

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

Revision	Date	Comments
Initial	Jan 22 2013	Initial draft version. Evgeni Grigorov, ProSyst Software, _e.grigorov@prosyst.com_

Revision	Date	Comments
2nd draft	Feb 13 2013	Updated Considered Alternatives and Security Considerations after F2F meeting in Austin, TX. Provide more details about device management. Evgeni Grigorov, ProSyst Software, *e.grigorov@prosyst.com*
3rd draft	Mar 08 2013	Remove DeviceAdmin service. Describe DeviceFunction and FunctionalDevice interfaces. Evgeni Grigorov, ProSyst Software, *e.grigorov@prosyst.com*
4th draft	Apr 08 2013	Rename the package and some constants. Merge the AbstractDevice and FunctionalDevice to FunctionalDevice. Add Functional Device Permission. Add Device Function Event. Minor fixes: renamed Device Access category, fixed unit representation and some clarifications. Add a suggestion about Device Functions to be discussed on F2F in Cologne. Evgeni Grigorov, ProSyst Software, *e.grigorov@prosyst.com*
5th draft	Jun 12 2013	Add a basic set of Device Functions. Include the device status transitions. Update the illustrations. Add a status detail mapping. Add some snippets. Remove the device helper methods for an access to parent, children and reference devices. Add a Functional Device and Device Function descriptions. Add error codes to DeviceFunctionException. Update the javadoc. Evgeni Grigorov, ProSyst Software, *e.grigorov@prosyst.com*
6th draft	Jul 02 2013	Describe the status transitions in detail. FunctionalDeviceException.CODE_UNKNOW fixed to CODE_UNKNOWN. Functional Group is introduced. Functional Device, Functional Group and Device Function are in the service registry. New service properties are introduced. Parent-child relation is removed. Add more details to the descriptions. Evgeni Grigorov, ProSyst Software, *e.grigorov@prosyst.com*

February 24, 2014

Revision	Date	Comments
7th draft	Sept 09 2013	Basic device function set is updated. Rename FunctionalDevice to Device. Rename FunctionalDeviceException to DeviceException. Rename FunctionalDevicePermission to DevicePermission. Relax the relation between the device and device function. DeviceExcpetion extends IOException. Functional group is removed. Renamed device function metadata properties. Evgeni Grigorov, ProSyst Software, *e.grigorov@prosyst.com*
8th draft	Jan 16 2014	Service property names are renamed form PROPERTY_<name> to SERVICE_<name>. Status disabled is removed, because it's applicable to small set of devices like peripherals. Remove the public methods to update the device properties. They should be initially configured. Updated permissions, because of updated device management operations. Overview diagram is added. Diagram with all device statuses is added. The package is renamed. Common device function data structure is introduced. Property and operation metadata structures are introduced. Device function type is added. There is a new interface with base set of device function types. There is a new interface with SI unit symbols. Evgeni Grigorov, ProSyst Software, *e.grigorov@prosyst.com*
9th draft	Jan 30 2014	Device.setName is removed. The device properties configuration is a vendor specific. Minor javadoc fixes and name improvements after the initial reference implementation. Device Function must be registered under only one interface. WakeUp Device Function is introduced to cover bettery-operated devices. Evgeni Grigorov, ProSyst Software, *e.grigorov@prosyst.com*
10th draft	Feb 21 2014	Based on REG F2F meeting in Cologne: Basic device functions are moved to another RFC document. DeviceFunction renamed to Function. DeviceFunctionEvent renamed to FunctionEvent. DeviceFunctionData renamed to FunctionData.

1   Introduction

OSGi is gaining popularity as enabling technology for building embedded system in residential and M2M markets. In these contexts it is often necessary to communicate with IP and non-IP devices by using various protocols such as ZigBee, Z-Wave, KNX, UPnP etc. In order to provide a convenient programming model suitable for the realization of end-to-end services it is very useful to define and apply an abstraction layer which unifies the work with devices supporting different protocols.

This RFC defines a new device abstraction API in OSGi.

2   Application Domain

Currently there are several standardization bodies such as OSGi Alliance, HGI, BBF, ETSI M2M which deal with the deployment of services in an infrastructure based on the usage of a Residential Gateway running OSGi as Execution Platform. The picture on Illustration 1 shows a reference architecture which is valid in the majority of cases under consideration.
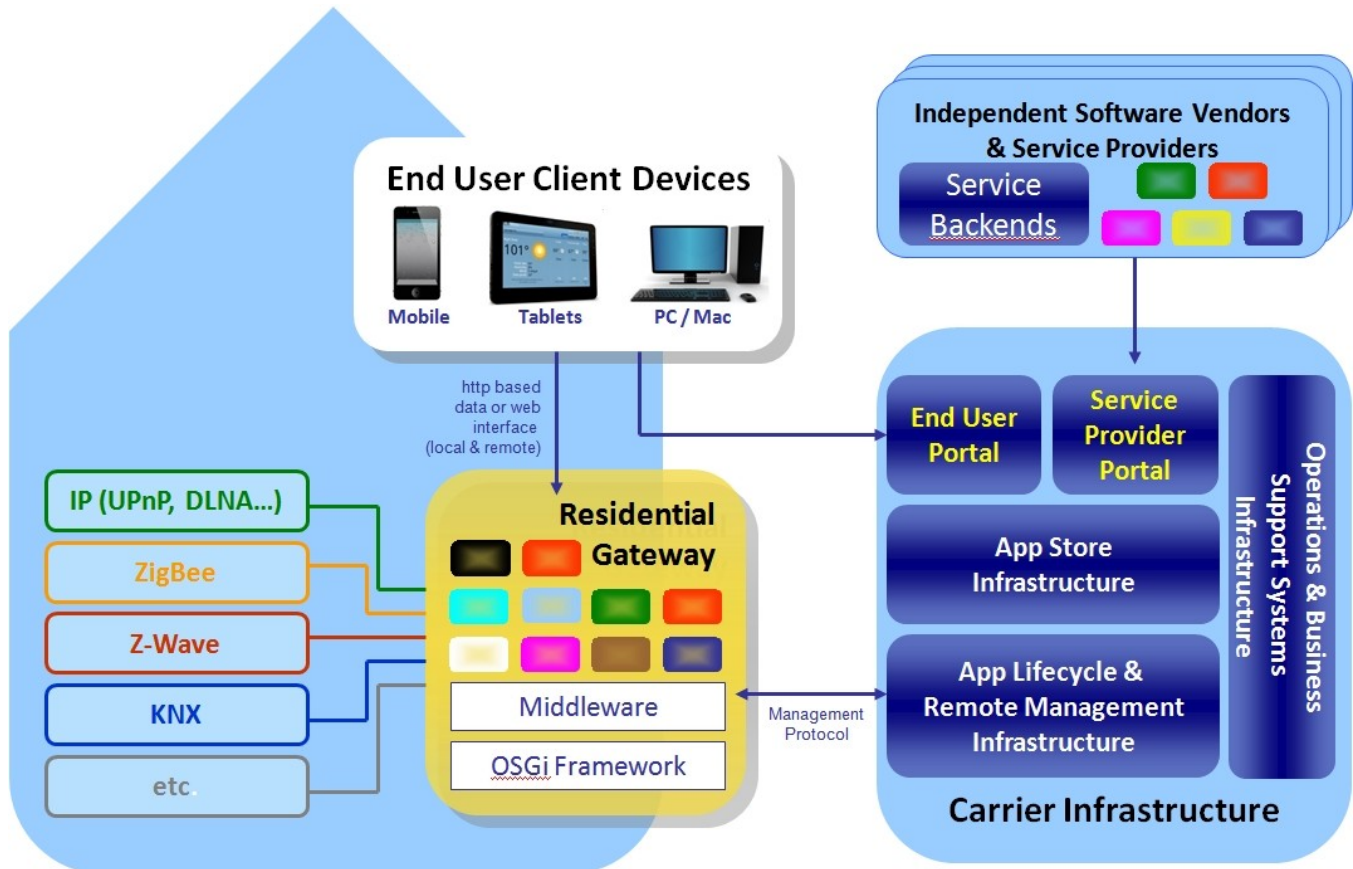
Illustration 1

In this architecture the application logic is distributed between:

- Applications running on the residential gateways

- Applications running in the cloud, e.g. on the service provider's backend

- Applications on the devices providing UI (e.g. tablets, mobile phones, desktops).

In order to realize services which access other IP and non-IP devices connected to the residential gateway, those applications must be able to read information from the devices and perform operations on them through software APIs. Such an access is essential for services in the area of smart metering, entertainment, home automation, assisted living and security.

The existing OSGi specifications which address related topics are:

- Device Access Specification – focuses on the dynamic discovery of the proper driver when a new device is attached/connected to the residential gateway. The device access is limited to attend the driver installation needs.

- UPnP™ Device Service Specification – defines among the other OSGi API for work with UPnP devices accessible from the residential gateway. API is specified in the scope of UPnP Device Access category.

3   Problem Description

Normally the residential gateways operate in heterogeneous environment including devices that support different protocols. It's not trivial to provide interoperability of the applications and the devices under such circumstances. The existing OSGi Device Access Specification solves the driver installation problems but currently there is no complete API that can be used for accessing the device data and for invoking actions on the devices.

Illustration 2 shows one possible approach for working with heterogeneous devices in an OSGi environment:
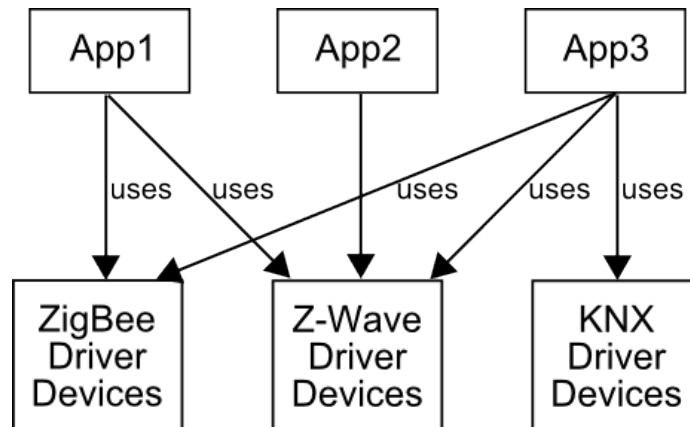
Illustration2

In this case each application which accesses devices of a given type must use API specific for this type. One obvious disadvantage of this model is that when a new device protocol is added the applications must be modified in order to support this protocol.

Much better is the approach from Illustration 3 which is defined by this RFC.
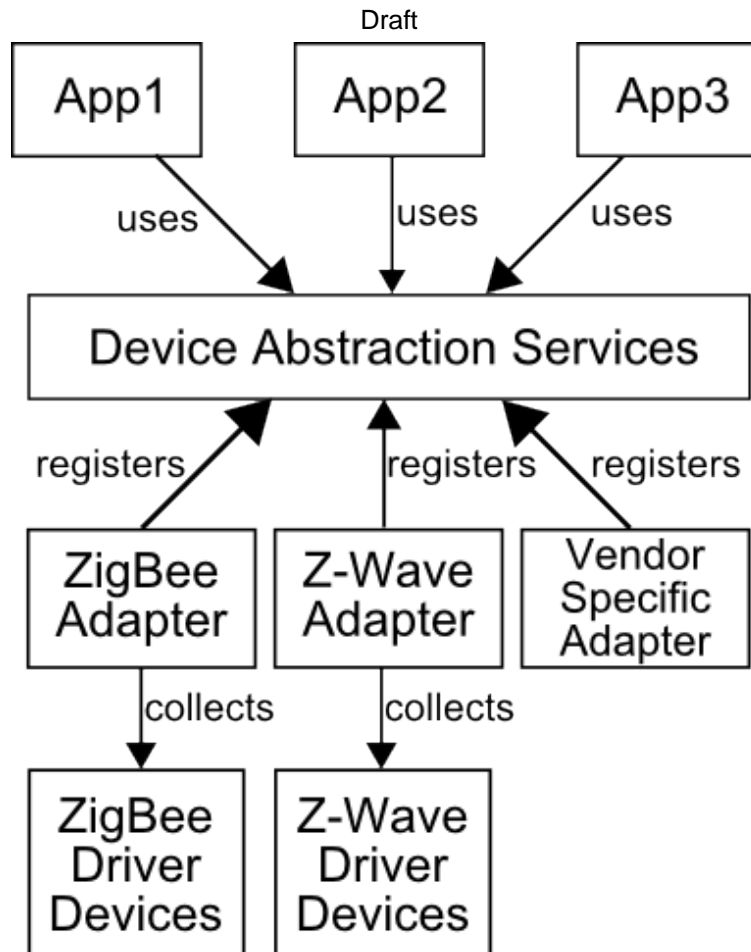
Illustration3

In this case an additional device abstraction layer is introduced which unifies the work with the devices provided by the different underlying protocols. Thus the following advantages are achieved:

- The application programmers can work with devices provided by different protocols exactly in the same way and by applying the same program interface. The protocol adapters and device abstraction API hide the complexity/differences of the device protocols.

- The applications can work without modification when new hardware controllers and protocol adapters are dynamically added.

- When remote access to the devices connected to the gateway is necessary (e.g. in m2m and management scenarios) it's much easier to provide mapping to one API then to a set of protocol dependent APIs.

- It is much easier to build UI for remote browsers or for apps running on mobile devices if just one mapping to one unified device abstraction API is necessary.

4 Requirements

Requirement 1. The solution MUST define API for controlling devices which is applicable for all relevant device protocols.

Requirement 2. The solution MUST define API for controlling devices which is independent from the device protocols.

Requirement 3. The solution MUST include device access control based on user and application permissions compliant with the OSGi security model.

Requirement 4. The solution MUST take advantage of the security features available in the device protocols.

Requirement 5. The solution MUST include a device protocol independent notification mechanism realized according to the OSGi event mechanisms.

Requirement 6. The solution SHOULD be mappable to other relevant standards such as HGI, ETSI M2M and BBF handling the remote access to device networks.

Requirement 7. The solution MUST provide configurable device data and metadata model.

Requirement 8. The solution MUST be applicable to the changeable device behavior. Sleeping/power saving devices can go and stay offline for a long time, but should be available in the defined API.

Requirement 9. The solution MUST provide an extension mechanism to support devices provided by new protocols.

Requirement 10. The solution MAY provide means to access the protocol specific device object.

Requirement 11. The solution MUST register device or/and device related instance to the OSGi service registry.

Requirement 12. The solution MAY update OSGi Device Access Specification.

5 Technical Solution

5.1 Introduction

Remote device control provides opportunity to save energy, to provide better security, to save your time during daily tasks and many more. The devices can play different roles in their networks as events reporters, controllers etc. That dynamic behavior is well mappable to the dynamic OSGi service registry. There is a registration of `Device` service. It realizes basic set of management operations and provides rich set of properties. The applications are allowed to track the device status, to read descriptive information and to follow the device relations. A set of functions can belong to the device. They represents the device operations and related properties in an atomic way. The device functions can be found in the OSGi service registry. The applications are

allowed to get directly the required functions if they don't need information about the device. For example, light device is registered as a `Device` service and there is a `Function` service to turn on and turn off the light.

5.1.1 Entities

- Device – represents the device in the OSGi service registry. It's described with a set of service properties and provides basic management operations.

- Function – atomic functional entity. The device can support a few functions like switch and sensor. The function provides a set of properties and operations.

- FunctionEvent – asynchronous event. It's sent through EventAdmin service and notifies for Function property change.

- FunctionData – data structure carries Function property value with additional metadata.

- PropertyMetadata and OperationMetadata – contains metadata about the Function properties and operations.
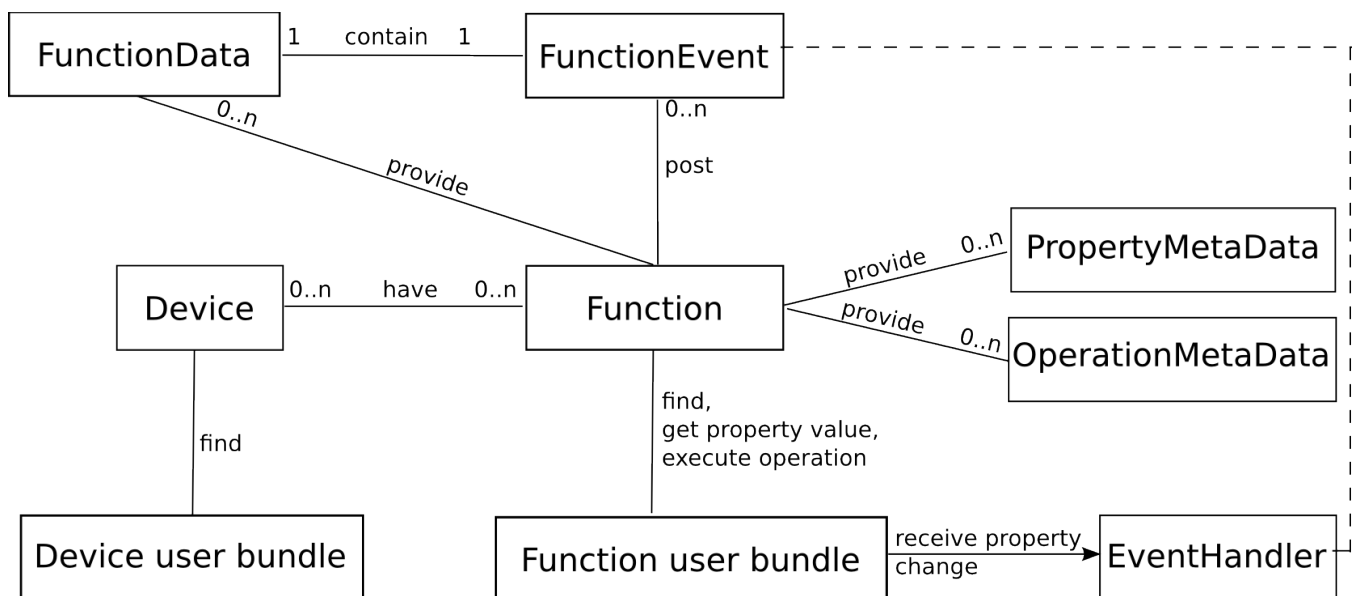
Illustration 4: Device Abstraction Layer Overview

5.2   Device Access Category

The device access category is called "DAL". The category name is defined as a value of `Device.DEVICE_CATEGORY` constant. It can be used as a part of `org.osgi.service.device.Constants.DEVICE_CATEGORY` service property key value. The category impose this specification rules.

5.3   Device Service

`Device` interface is dedicated for a common access to the devices provided by different protocols. It can be mapped one to one with the physical device, but can be mapped only with a given functional part of the device. In this scenario, the physical device can be realized with a set of `Device` services and different relations between them. `Device` service can represent pure software unit. For example, it can simulate the real device work. There are basic management operations for remove, property access and property update. New protocol devices can be supported with a registration of new `Device` services.

If the underlying protocol and the implementation allow, the `Device` services must be registered again after the OSGi framework reboot. The service properties must be restored, the supported functions must be provided and `Device` relations must be visible to the applications.

The OSGi service registry has the advantage of being easily accessible. The services can be filtered and accessed with their properties. The device service has a rich set of such properties as it is on Illustration 5:

- `Device.SERVICE_UID` – Specifies the device unique identifier. It's a mandatory property. The value type is `java.lang.String`. To simplify the unique identifier generation, the property value must follow the rule:

  UID ::= driver-name ':' device-id

  UID - device unique identifier

  driver-name - the value of the `Device.SERVICE_DRIVER` service property

  device-id - device unique identifier in the scope of the driver

- `Device.SERVICE_REFERENCE_UIDS` – Specifies the reference device unique identifiers. It's an optional property. The value type is `java.lang.String[]`. It can be used to represent different relationships between the devices. For example, The ZigBee controller can have a reference to the USB dongle.

- `Device.SERVICE_DRIVER` – Specifies the device driver name. For example, ZigBee, Z-Wave, Bluetooth etc. It's a mandatory property. The value type is `java.lang.String`.

- `Device.SERVICE_NAME` – Specifies the device name. It's an optional property. The value type is `java.lang.String`.

- `Device.SERVICE_STATUS` – Specifies the current device status. It's a mandatory property. The value type `java.lang.Integer`. The possible values are:

  - `Device.STATUS_REMOVED` – Indicates that the device is removed from the network. That status must be set as the last device status and after that the device service can be unregistered from the service registry. The status is available for stale device services too. All transitions to and from this status are described in Transitions to STATUS_REMOVED section.

  - `Device.STATUS_OFFLINE` – Indicates that the device is currently not available for operations. The end device is still installed in the network and can become online later. The controller is unplugged or there is no connection. All transitions to and from this status are described in detail in Transitions to and from STATUS_OFFLINE section.

  - `Device.STATUS_ONLINE` – Indicates that the device is currently available for operations. All transitions to and from this status are described in detail in Transitions to and from STATUS_ONLINE section.

  - `Device.STATUS_PROCESSING` – Indicates that the device is currently busy with an operation. All transitions to and from this status are described in detail in Transitions to and from STATUS_PROCESSING section.

  - `Device.STATUS_NOT_INITIALIZED` – Indicates that the device is currently not initialized. Some protocols don't provide device information right after the device is connected. The device can be initialized later when it's awakened. All transitions to and from this status are described in detail in Transitions to and from STATUS_NOT_INITIALIZED section.

  - `Device.STATUS_NOT_CONFIGURED` – Indicates that the device is currently not configured. The device can require additional actions to become completely connected to the network. All transitions to and from this status are described in detail in Transitions to and from STATUS_NOT_CONFIGURED section.

- `Device.SERVICE_STATUS_DETAIL` – Provides the reason for the current device status. It's an optional property. The property value cannot be externally set or modified. The value type is `java.lang.Integer`. There are two value categories. Positive values indicate the reason for the current status like `Device.STATUS_DETAIL_CONNECTING`. Negative values indicate errors related to the current device status like `Device.STATUS_DETAIL_DEVICE_BROKEN`. The list with defined status details is:

    - `Device.STATUS_DETAIL_CONNECTING` – The reason for the current device status is that the device is currently connecting to the network. It indicates the reason with a positive value `1`. The device status must be `STATUS_PROCESSING`.

    - `Device.STATUS_DETAIL_INITIALIZING` – The reason for the current device status is that the device is currently in process of initialization. It indicates the reason with a positive value `2`. The network controller initializing means that information about the network is currently read. The device status must be `STATUS_PROCESSING`.

    - `Device.STATUS_DETAIL_REMOVING` – The reason for the current device status is that the device is leaving the network. It indicates the reason with positive value 3. The device status must be `STATUS_PROCESSING`.

    - `Device.STATUS_DETAIL_CONFIGURATION_NOT_APPLIED` – The reason for the current device status is that the device configuration is not applied. It indicates an error with a negative value `-1`. The device status must be `STATUS_NOT_CONFIGURED`.

    - `Device.STATUS_DETAIL_DEVICE_BROKEN` – The reason for the offline device is that the device is broken. It indicates an error with a negative value `-2`. The device status must be `STATUS_OFFLINE`.

    - `Device.STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR` – The reason for the current device status is that the device communication is problematic. It indicates an error with a negative value `-3`. The device status must be `STATUS_ONLINE` or `STATUS_NOT_INITIALIZED`.

    - `Device.STATUS_DETAIL_DEVICE_DATA_INSUFFICIENT` – The reason for the uninitialized device is that the device doesn't provide enough information and cannot be determined. It indicates an error with a negative value `-4`. The device status must be `STATUS_NOT_INITIALIZED`.

    - `Device.STATUS_DETAIL_DEVICE_NOT_ACCESSIBLE` – The reason for the offline device is that the device is not accessible and further communication is not possible. It indicates an error with a negative value `-5`. The device status must be `STATUS_OFFLINE`.

    - `Device.STATUS_DETAIL_ERROR_APPLYING_CONFIGURATION` – The reason for the current device status is that the device cannot be configured. It indicates an error with a negative value `-6`. The device status must be `STATUS_NOT_CONFIGURED`.

    - `Device.STATUS_DETAIL_IN_DUTY_CYCLE` – The reason for the offline device is that the device is in duty cycle. It indicates an error with a negative value `-7`. The device status must be `STATUS_OFFLINE`.

Custom status details are allowed, but they must not overlap the specified codes. Table 1 contains the mapping of the status details to the statuses.

Status Detail	Status
STATUS_DETAIL_CONNECTING	STATUS_PROCESSING
STATUS_DETAIL_INITIALIZING	STATUS_PROCESSING
STATUS_DETAIL_REMOVING	STATUS_PROCESSING
STATUS_DETAIL_CONFIGURATION_NOT_APPLIED	STATUS_NOT_CONFIGURED
STATUS_DETAIL_DEVICE_BROKEN	STATUS_OFFLINE
STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR	STATUS_ONLINE, STATUS_NOT_INITIALIZED
STATUS_DETAIL_DEVICE_DATA_INSUFFICIENT	STATUS_NOT_INITIALIZED
STATUS_DETAIL_DEVICE_NOT_ACCESSIBLE	STATUS_OFFLINE
STATUS_DETAIL_ERROR_APPLYING_CONFIGURATION	STATUS_NOT_CONFIGURED
STATUS_DETAIL_IN_DUTY_CYCLE	STATUS_OFFLINE

Table 1

- `Device.SERVICE_HARDWARE_VENDOR` – Specifies the device hardware vendor. It's an optional property. The value type is `java.lang.String`.

- `Device.SERVICE_HARDWARE_VERSION` – Specifies the device hardware version. It's an optional property. The value type is `java.lang.String`.

- `Device.SERVICE_FIRMWARE_VENDOR` – Specifies the device firmware vendor. It's an optional property. The value type is `java.lang.String`.

- `Device.SERVICE_FIRMWARE_VERSION` – Specifies the device firmware version. It's an optional property. The value type is `java.lang.String`.

- `Device.SERVICE_TYPES` – Specified the device types. It's an optional property. The value type is `java.lang.String[]`.

- `Device.SERVICE_MODEL` – Specifies the device model. It's an optional property. The value type is `java.lang.String`.

- `Device.SERVICE_SERIAL_NUMBER` – Specifies the device serial number. It's an optional property. The value type is `java.lang.String`.

The device services are registered in the OSGi service registry with `org.osgi.services.functionaldevice.Device` interface. The next code snippet prints the online devices.

```
final ServiceReference[] deviceSRefs = context.getServiceReferences(
  Device.class.getName(),
  '(' + Device.SERVICE_STATUS + '=' + Device.STATUS_ONLINE + ')');
if (null == deviceSRefs) {
  return; // no such services
}
for (int i = 0; i < deviceSRefs.length; i++) {
```

```
    printDevice(deviceSRefs[i]);
}
```
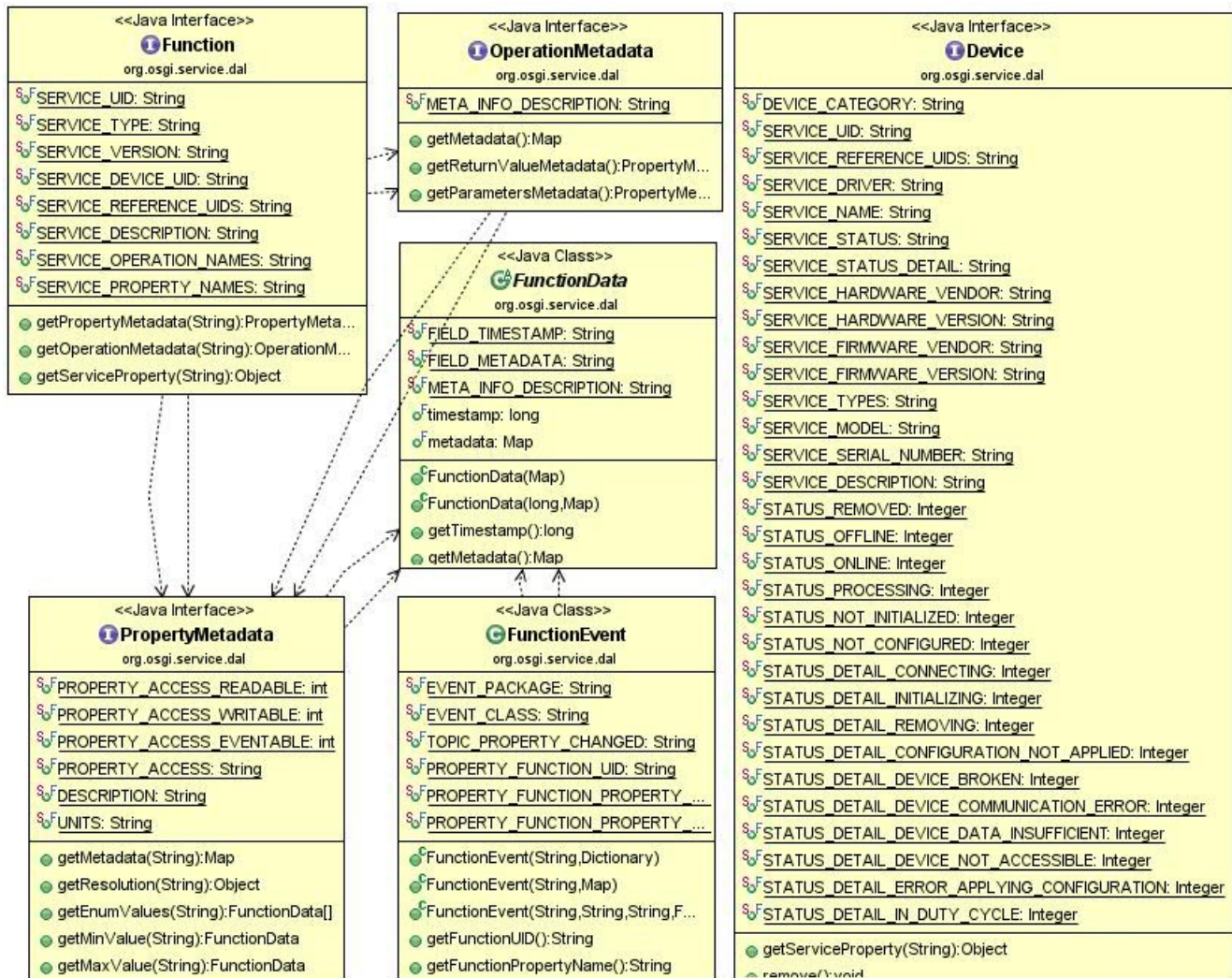

Illustration5

Applications need to have an access to the device properties. For convenience there is a helper method:

- `getServiceProperty(String propName)` – Returns the current value of the specified property. The method will return the same value as `org.osgi.framework.ServiceReference.getProperty(String)` for the service reference of this device.

5.3.1 Reference Device Services

`Device` service can have a reference to other devices. That link can be used to represent different relationships between devices. For example, the ZigBee dongle can be used as USB `Device` and ZigBee network controller

`Device`. The network controller device can have a reference to the physical USB device as it's depicted on Illustration 6.

The related service property is `Device.SERVICE_REFERENCE_UIDS`.
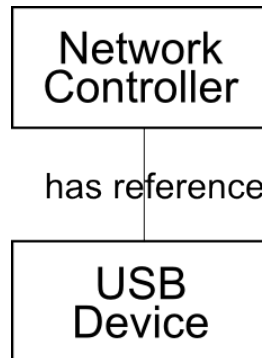
Illustration6

5.3.2 Device Service Registration

The devices are registered as services in the OSGi service registry. The service interface is `org.osgi.services.functionaldevice.Device`. There is a registration order. `Device` services are registered last. Before their registration, there is `Function` service registration.

5.3.3 Device Service Unregistration

OSGi service registry is only about the read-only access for the services. There are no control operations. The service provider is responsible to register, update or unregister the services. That design is not very convenient for the device life cycle. The `Device` interface provides a callback method `remove()`. The method can be optionally implemented by the device provider. `java.lang.UnsupportedOperationException` can be thrown if the method is not supported. When the remove callback is called, an appropriate command will be synchronously send to the device. As a result it can leave the network and device related service will be unregistered. There is an unregistration order. The registration reverse order is used when the services are unregistered. `Device` services are unregistered first before `Function` services.

5.4 Device Status Transitions

The device status uncover the device availability. It can demonstrate that device is currently not available for operations or that the device requires some additional configuration steps. The status can jump over the different values according to the rules defined in this section. The status transitions are summarized in Table 2, visualized in Illustration 7 and described in detail in the next sections. The entry device status is always `STATUS_PROCESSING`. When the device info is processed, the device can go to another status. The last possible device status is `STATUS_REMOVED`. The status must be set when the device is removed from the network. After that status, the device service will be unregistered.
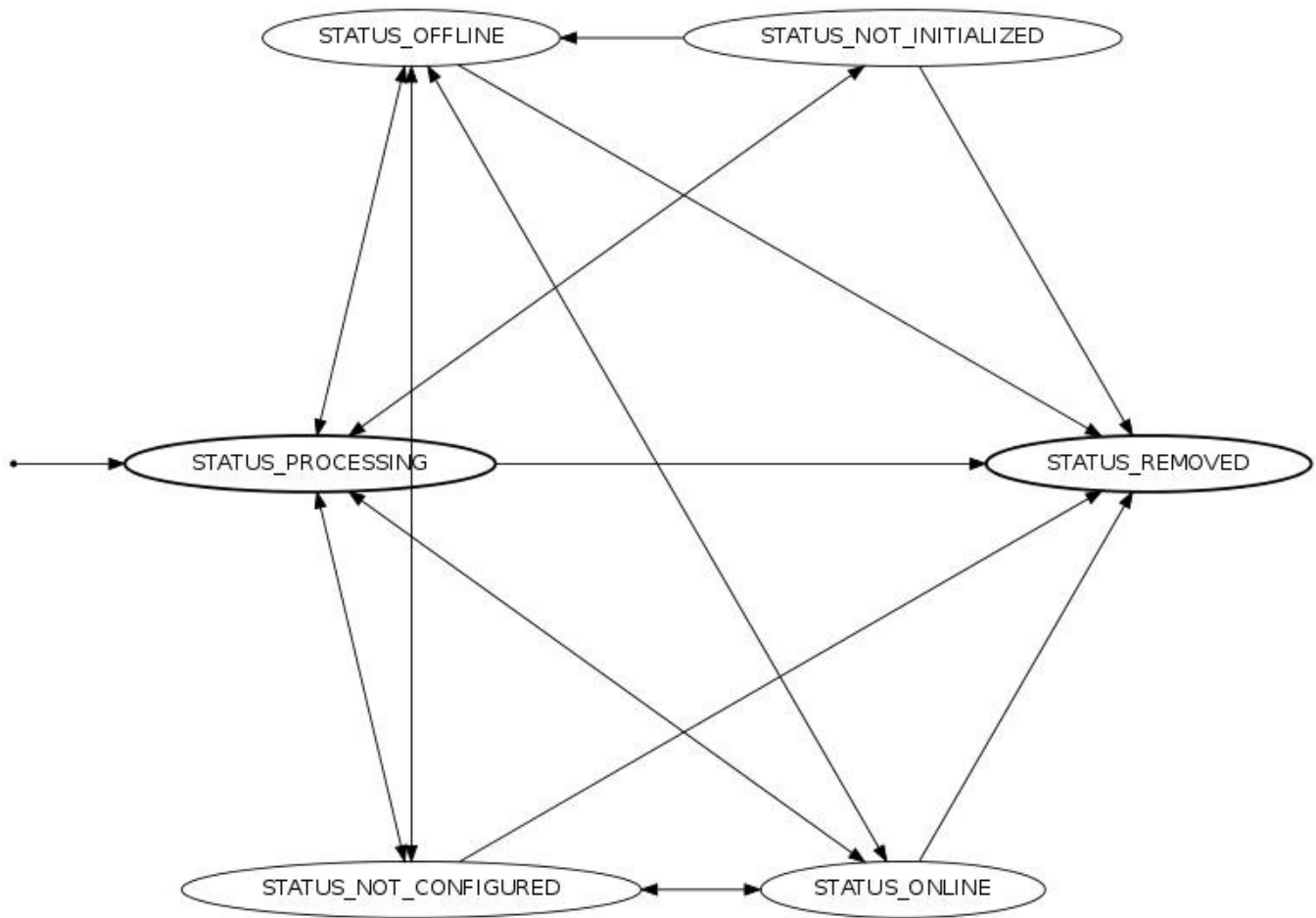
Illustration7

From \ To Status	PROCESSING	ONLINE	OFFLINE	NOT_INITIALIZED	NOT_CONFIGURED	REMOVED
PROCESSING	-	Initial device data has been read.	Device is not accessible.	Initial device data is partially read.	Device has a pending configuration.	Device is removed.
ONLINE	Device data is processing.	-	Device is not accessible.	-	Device has a new pending configuration.	Device is removed.
OFFLINE	Device data is processing.	Device data has been read.	-	-	Device has a pending configuration.	Device is removed.
NOT_INITIALIZED	Device data is processing.	-	Device is not accessible.	-	-	Device is removed.
NOT_CONFIGURED	Device data is processing.	Device pending configuration is satisfied.	Device is not accessible.	-	-	Device is removed.
REMOVED	-	-	-	-	-	-

Table2

5.4.1 Transitions to STATUS_REMOVED

The device can go to `Device.STATUS_REMOVED` from any other status. Once reached, the device status cannot be updated any more. The device is removed from the network and the device service is unregistered from the OSGi service registry. If there are stale references to the `Device` service, their status will be set to `STATUS_REMOVED`.

The common way for a given device to be removed is `Device.remove()`. When the method returns, the device status will be `STATUS_REMOVED`. It requires a synchronous execution of the operation.

5.4.2 Transitions to and from STATUS_OFFLINE

The `STATUS_OFFLINE` indicates that the device is currently not available for operations. That status can be set, because of different reasons. The network controller can be unplugged, connection to the device is lost etc. This variety provides an access to that status from any other except `STATUS_REMOVED`. Transitions to and from this status are:

• From `STATUS_OFFLINE` to `STATUS_REMOVED` – device is removed. The status can be set as a result of `Device.remove()` method call.

• From `STATUS_OFFLINE` to `STATUS_PROCESSING` – device data is processing.

• From `STATUS_OFFLINE` to `STATUS_NOT_CONFIGURED` – device has a pending configuration.

• From `STATUS_OFFLINE` to `STATUS_ONLINE` – device data has been read and the device is currently available for operations.

• From `STATUS_OFFLINE` to `STATUS_NOT_INITIALIZED` – That transition is not possible, because the status have to go through `STATUS_PROCESSING`. If the processing is unsuccessful, `STATUS_NOT_INITIALIZED` will be set.

- To STATUS_OFFLINE from STATUS_REMOVED – That transition is not possible. If device is removed, the service will be unregistered from the service registry.

- To STATUS_OFFLINE from STATUS_PROCESSING – device is not accessible any more while device data is processing.

- To STATUS_OFFLINE from STATUS_NOT_CONFIGURED – Not configured device is not accessible any more.

- To STATUS_OFFLINE from STATUS_ONLINE – Online device is not accessible any more.

- To STATUS_OFFLINE from STATUS_NOT_INITIALIZED – Not initialized device is not accessible any more.

The possible transitions are summarized on Illustration 8.

Illustration8

5.4.3 Transitions to and from STATUS_ONLINE

The STATUS_ONLINE indicates that the device is currently available for operations. The online devices are initialized and ready for use. Transitions to and from this status are:
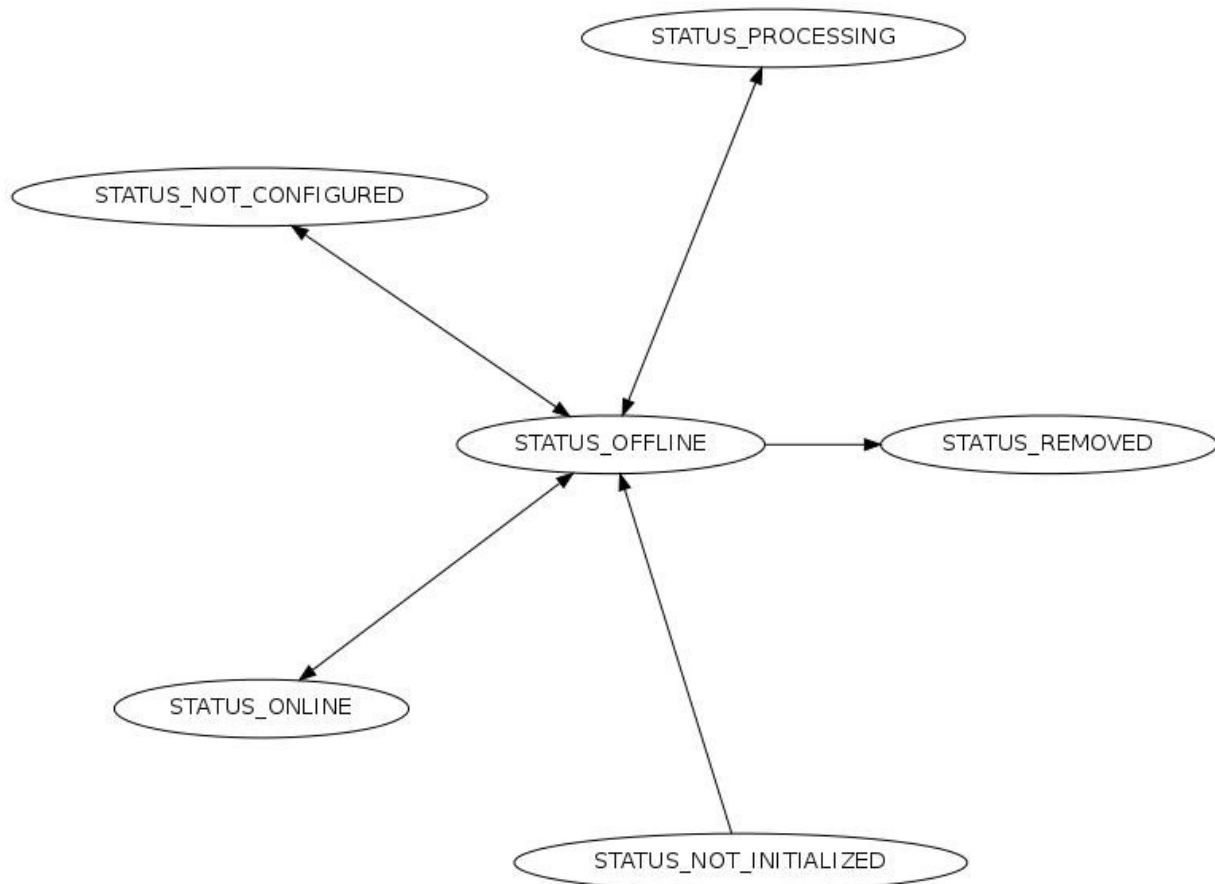
- From STATUS_ONLINE to STATUS_REMOVED – device is removed. The status can be set as a result of Device.remove() method call.

- From STATUS_ONLINE to STATUS_PROCESSING – device data is processing.

- From `STATUS_ONLINE` to `STATUS_NOT_CONFIGURED` – device has a pending configuration.

- From `STATUS_ONLINE` to `STATUS_OFFLINE` – Online device is not accessible any more.

- From `STATUS_ONLINE` to `STATUS_NOT_INITIALIZED` – That transition is not possible. Online devices are initialized.

- To `STATUS_ONLINE` from `STATUS_REMOVED` – That transition is not possible. If device is removed, the service will be unregistered from the service registry.

- To `STATUS_ONLINE` from `STATUS_PROCESSING` – Initial device data has been read. The device is available for operations.

- To `STATUS_ONLINE` from `STATUS_NOT_CONFIGURED` – The device pending configuration is satisfied.

- To `STATUS_ONLINE` from `STATUS_OFFLINE` – device is accessible for operations.

- To `STATUS_ONLINE` from `STATUS_NOT_INITIALIZED` – That transition is not possible. The device data has to be processed and then the device can become online. Intermediate status `STATUS_PROCESSING` will be used.

The possible transitions are summarized on Illustration 9.

Illustration9

5.4.4 Transitions to and from STATUS_PROCESSING

The status indicates that the device is currently busy with an operation. It can be time consuming operation and can result to any other status. The operation processing can be reached by any other status except `STATUS_REMOVED`. An example, offline device requires some data processing to become online. It will apply the statuses `STATUS_OFFLINE`, `STATUS_PROCESSING` and `STATUS_ONLINE`. Transitions to and from this status are:
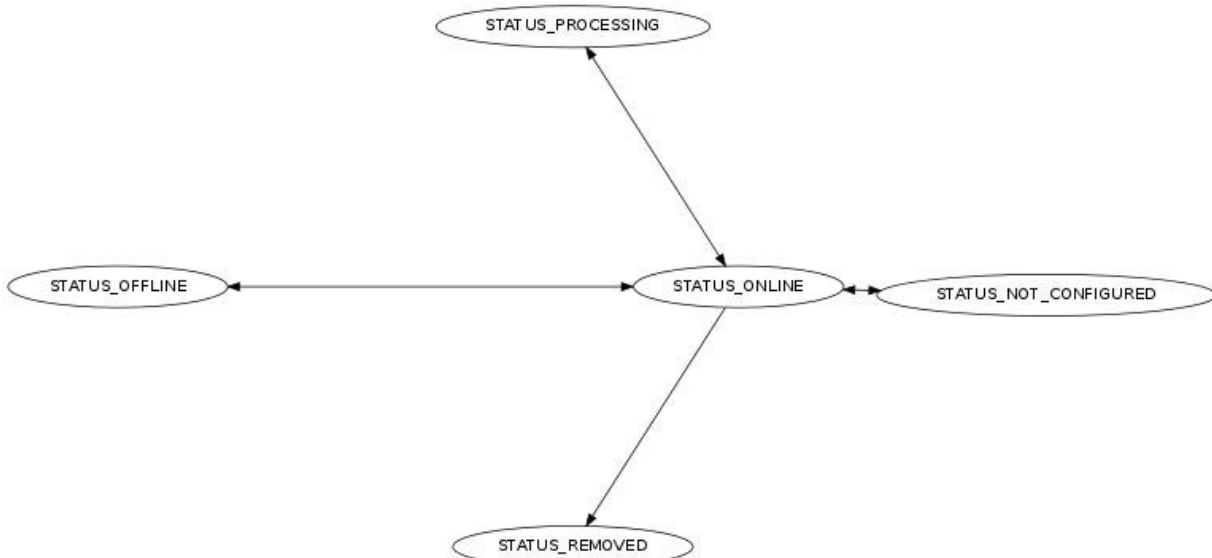
- From `STATUS_PROCESSING` to `STATUS_REMOVED` – device is removed. The status can be set as a result of `Device.remove()` method call.

- From `STATUS_PROCESSING` to `STATUS_ONLINE` – Initial device data has been read. The device is available for operations.

- From `STATUS_PROCESSING` to `STATUS_NOT_CONFIGURED` – device has a pending configuration.

- From STATUS_PROCESSING to STATUS_OFFLINE – Online device is not accessible any more.

- From STATUS_PROCESSING to STATUS_NOT_INITIALIZED – device initial data is partially read.

- To STATUS_PROCESSING from STATUS_REMOVED – That transition is not possible. If device is removed, the service will be unregistered from the service registry.

- To STATUS_PROCESSING from STATUS_ONLINE – device is busy with an operation.

- To STATUS_PROCESSING from STATUS_NOT_CONFIGURED – The device pending configuration is satisfied and the device is busy with an operation.

- To STATUS_PROCESSING from STATUS_OFFLINE – device is busy with an operation.

- To STATUS_PROCESSING from STATUS_NOT_INITIALIZED – device initial data is processing.

The possible transitions are summarized on Illustration 10.

Illustration 10

5.4.5 Transitions to and from STATUS_NOT_INITIALIZED

The status indicates that the device is currently not initialized. Some protocols don't provide device information right after the device is connected. The device can be initialized later when it's awakened. Not initialized device requires some data processing to become online. STATUS_PROCESSING is used as an intermediate status. Transitions to and from this status are:

- From STATUS_NOT_INITIALIZED to STATUS_REMOVED – device is removed. The status can be set as a result of Device.remove() method call.

- From STATUS_NOT_INITIALIZED to STATUS_PROCESSING – device data is processing.

- From STATUS_NOT_INITIALIZED to STATUS_NOT_CONFIGURED – That transition is not possible. device requires some data processing.

- From STATUS_NOT_INITIALIZED to STATUS_OFFLINE – device is not accessible any more.

- From STATUS_NOT_INITIALIZED to STATUS_ONLINE – That transition is not possible. Device requires some data processing to become online.

- To `STATUS_NOT_INITIALIZED` from `STATUS_REMOVED` – That transition is not possible. If device is removed, the service will be unregistered from the service registry.

- To `STATUS_NOT_INITIALIZED` from `STATUS_PROCESSING` – device data is partially read.

- To `STATUS_NOT_INITIALIZED` from `STATUS_NOT_CONFIGURED` – That transition is not possible. When device pending configuration is satisfied, the device requires additional data processing.

- To `STATUS_NOT_INITIALIZED` from `STATUS_OFFLINE` – That transition is not possible. Device requires some data processing and then can become not initialized.

- To `STATUS_NOT_INITIALIZED` from `STATUS_ONLINE` – That transition is not possible. Online device is initialized.

The possible transitions are summarized on Illustration 11.
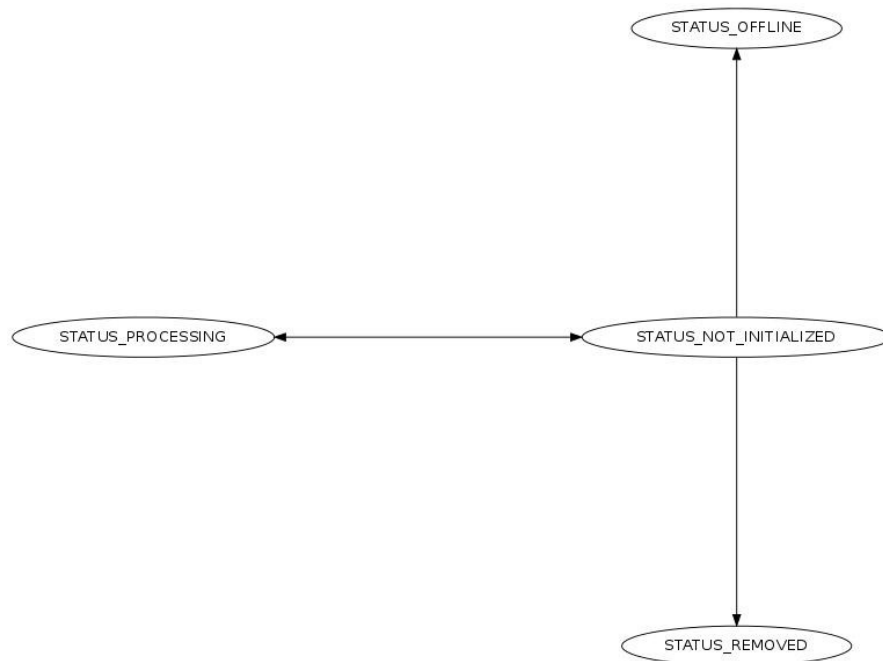
Illustration 11

5.4.6 Transitions to and from STATUS_NOT_CONFIGURED

Indicates that the device is currently not configured. The device can require additional actions to become completely connected to the network. For example, a given device button has to be pushed. That status doesn't have transitions with `STATUS_NOT_INITIALIZED`, because some data processing is required. Transitions to and from this status are:
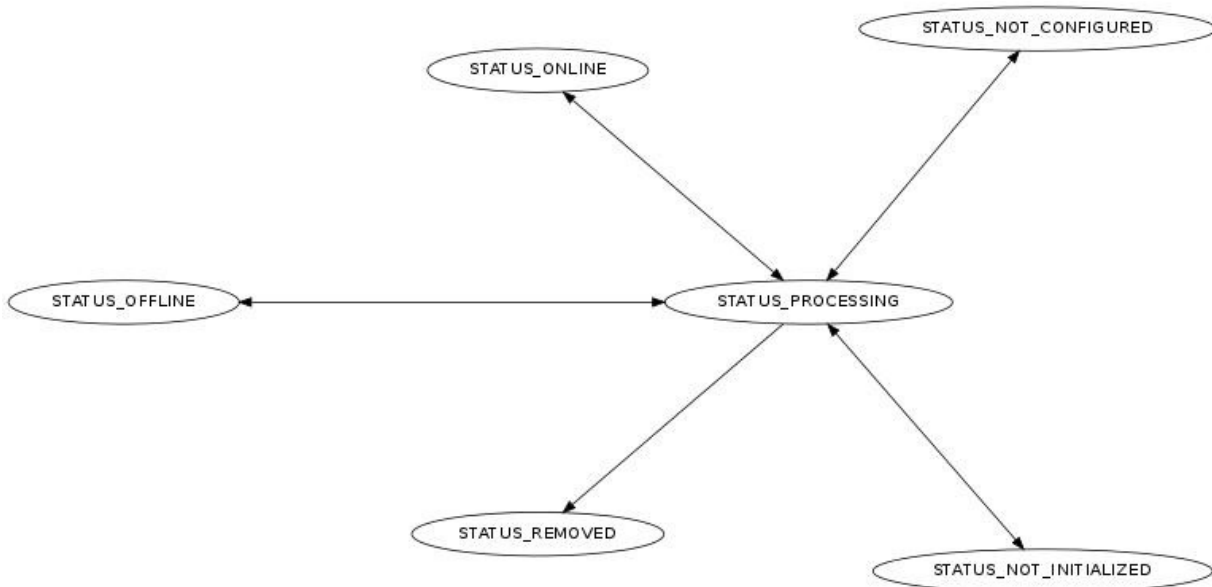
- From `STATUS_NOT_CONFIGURED` to `STATUS_REMOVED` – device is removed. The status can be set as a result of `Device.remove()` method call.

- From `STATUS_NOT_CONFIGURED` to `STATUS_PROCESSING` – device pending configuration is satisfied and some additional data processing is required.

- From `STATUS_NOT_CONFIGURED` to `STATUS_ONLINE` – device pending configuration is satisfied.

- From `STATUS_NOT_CONFIGURED` to `STATUS_OFFLINE` – device is not accessible any more.

- From `STATUS_NOT_CONFIGURED` to `STATUS_NOT_INITIALIZED` – That transition is not possible. When device pending configuration is satisfied, the device requires additional data processing.

- To `STATUS_NOT_CONFIGURED` from `STATUS_REMOVED` – That transition is not possible. If device is removed, the service will be unregistered from the service registry.

- To `STATUS_NOT_CONFIGURED` from `STATUS_PROCESSING` – Initial device data has been read but there is a pending configuration.

- To `STATUS_NOT_CONFIGURED` from `STATUS_ONLINE` – device has a pending configuration.

- To `STATUS_NOT_CONFIGURED` from `STATUS_OFFLINE` – device is going to be online, but has a pending configuration.

- To `STATUS_NOT_CONFIGURED` from `STATUS_NOT_INITIALIZED` – That transition is not possible. That transition is not possible. Device requires some data processing.

The possible transitions are summarized on Illustration 12.
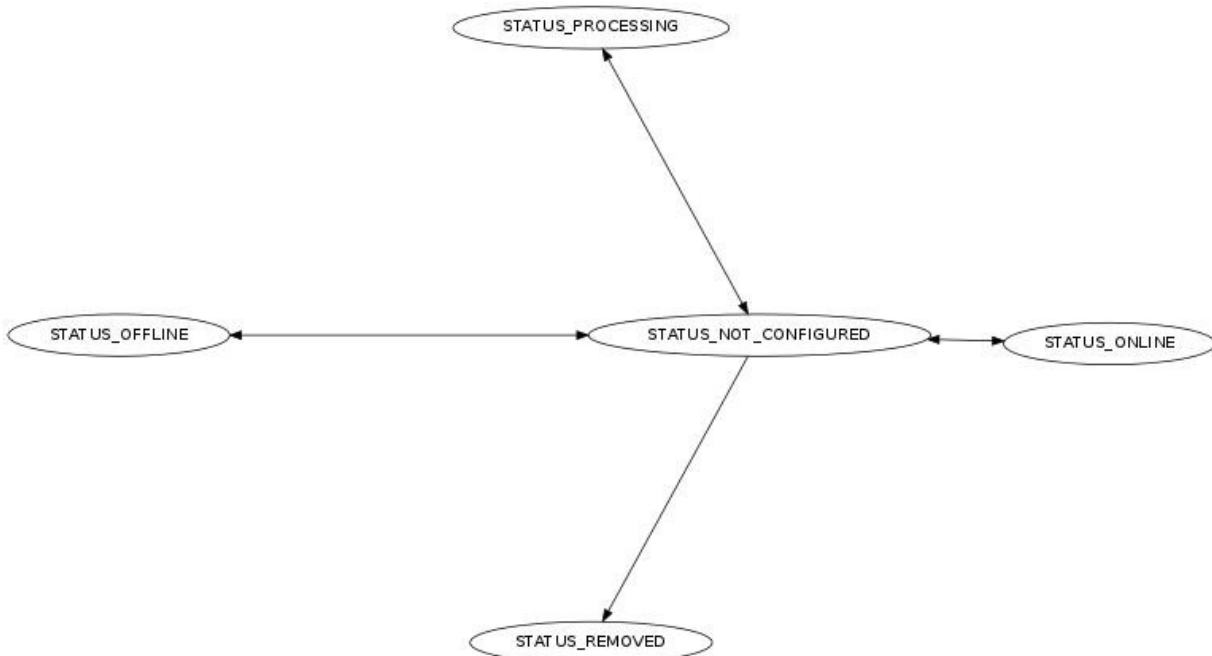
Illustration 12

5.5   Functions

The user applications can execute the device operations and manage the device properties. That control is realized with the help of `Function` services. The `Function` service can be registered in the service registry with those service properties:

- `Function.SERVICE_UID` – mandatory service property. The property value is the function unique identifier. `The value type is java.lang.String.` To simplify the unique identifier generation, the property value must follow the rule:

   function UID ::= device-id ':' function-id

   function UID – function unique identifier

   device-id – the value of the `Device.SERVICE_UID Device` service property

   function-id – function identifier in the scope of the device

If the function is not bound to a device, the function unique identifier can be device independent.

- `Function.SERVICE_TYPE` – mandatory service property. The service property value contains the function type. For example, the sensor function can have different types like temperature or pressure etc. It's an optional property. The value type is `java.lang.String`.

  Organizations that want to use function types that do not clash with OSGi Alliance defined types should prefix their types in own namespace.

- `Function.SERVICE_VERSION` – optional service property. The service property value contains the function version. That version can point to specific implementation version and vary in the different vendor implementations. The value type is `java.lang.String`.

- `Function.SERVICE_DEVICE_UID` – optional service property. The property value is the device identifier. The function belongs to this device. The value type is `java.lang.String`.

- `Function.SERVICE_REFERENCE_UIDS` – optional service property. The service property value contains the reference function unique identifiers. The value type is `java.lang.String[]`. It can be used to represent different relationships between the functions.

- `Function.SERVICE_DESCRIPTION` – optional service property. The property value is the function description. The value type is `java.lang.String`.
- `Function.SERVICE_OPERATION_NAMES` – optional service property. The property value is the function operation names. The value type is `java.lang.String[]`. It's not possible to exist two or more function operations with the same name i.e. the operation overloading is not allowed.
- `Function.SERVICE_PROPERTY_NAMES` – optional service property. The property value is the function property names. The value type is `java.lang.String[]`. It's not possible to exist two or more function properties with the same name.

The `Function` services are registered before the `Device` service. It's possible that `Function.SERVICE_DEVICE_UID` points to missing services at the moment of the registration. The reverse order is used when the services are unregistered. `Device` service is unregistered before the `Function` services.

Function service must be registered only under concrete function class. It's not allowed to register function service under more than one class. For example, those registrations are not allowed:

- `context.registerService(ManagedService.class.getName(), this, regProps);` - `ManagedService` interface is not a function interface.

- `context.registerService(Function.class.getName(), this, regProps);` - `Function` interface is not concrete function interface.

- `context.registerService(new String[] {BooleanControl.class.getName(), BooleanControl.class.getName()}, this, regProps);` - more than one function is used.

That one is a valid registration: `context.registerService(Meter.class.getName(), this, regProps);`. `Meter` is concrete function interface.

That rule helps to the applications to find the supported function class and to identify the metadata. Otherwise the function services can be accesses, but it's not clear which are the function classes and metadata.

5.5.1 Function Interface

Function is built by a set of properties and operations. The function can have unique identifier, type, version, description, link to the `Device` service and information about the reference functions. `Function` interface must be the base interface for all functions. If the device provider defines custom functions, all of them must extend `Function` interface. It provides a common access to the operations and properties meta data.

There are some general type rules, which unifies the access to the function data. They make easier the transfer over different protocols. All properties and operation arguments must use:

- Java primitive type or corresponding reference type.

- `java.lang.String`

- Java Beans, but their properties must use those rules. Java Beans are defined in JavaBeans specification [3].

- `java.util.Map` instances. The map keys can be any reference type of Java primitive types or `java.lang.String`. The values must use those rules.

- Arrays of defined types.

In order to provide common behavior, all functions must follow a set of common rules related to the implementation of their setters, getters, operations and events:

- The setter method must be executed synchronously. If the underlying protocol can return response to the setter call, it must be awaited. It simplifies the property value modifications and doesn't require asynchronous callback.

- The operation method must be executed synchronously. If the underlying protocol can return an operation confirmation or response, they must be awaited. It simplifies the operation execution and doesn't require asynchronous callback.

- The getter must return the last know cached property value. The device implementation is responsible to keep that value up to date. It'll speed up the applications when the function property values are collected. The same cached value can be shared between a few requests instead of a few calls to the real device.

- If a given function operation, getter or setter is not supported, `java.lang.UnsupportedOperationException` must be thrown. It indicates that function is partially supported.

- The function operations, getters and setters must not override `java.lang.Object` and this interface methods. For example:

    - `hashCode()` – it's `java.lang.Object` method and invalid function operation;

    - `wait()` – it's `java.lang.Object` method and invalid function operation;

    - `getClass()` – it's `java.lang.Object` method and invalid function getter;

    - `getPropertyMetadata(String propertyName)` – it's `org.osgi.service.dal.Function` method and invalid function getter.

5.5.2 Function Operations

`Function` operations are general callable units. They can perform a specific task on the device like turn on or turn off. They can be used by the applications to control the device. Operation names are available as a value of the service property `Function.SERVICE_OPERATION_NAMES`. The operations are identified by their names. It's not possible to exist two operations with the same name i.e. overloaded operations are not allowed or to override the property accessor methods. The operations are regular java methods. That implies that they have zero or more arguments and zero or one return value. The operation arguments and return value must follow the general type rules.

The operations can be optionally described with a set of meta data properties. Metadata is accessible with `Function.getOperationMetadata(String)` method. The result provides metadata about the operation, operation arguments and result value. Operation arguments and result value are using the same metadata as the function properties. The full details are defined in the next section.

5.5.3 Function Properties

`Function` properties are class fields. Their values can be read with getter methods and can be set with setter methods. The property names are available as a value of the service property `Function.SERVICE_PROPERTY_NAMES`. The properties are identified by their names. It's not possible to exist two properties with the same name.

The function properties must be integrated according to these rules:

- Getter methods must be available for all properties with `PropertyMetadata.PROPERTY_ACCESS_READABLE` access.

- Getter method must return a subclass of FunctionData.

- Setter methods must be available for all properties with `PropertyMetadata.PROPERTY_ACCESS_WRITABLE` access.

- Setter method must use `FunctionData` wrapped type. For example, there is `MyFunctionData` with timestamp, unit and `BigDecimal` value. The setter must accept as an argument the value of type `BigDecimal`.

- It's possible to have a second setter method, which accepts the value as a first argument and the unit as a second argument.

- No methods are required for properties with `PropertyMetadata.PROPERTY_ACCESS_EVENTABLE` access.

The accessor method names must be defined according JavaBeans specification [3].

The properties can be optionally described with a set of meta data properties. The property values can be collected with `Function.getPropertyMetadata(String)` method. The method result is `PropertyMetadata` with:

- Minimum value – available through `PropertyMetadata.getMinValue(String)`. The minimum value can be different for the different units.

- Maximum value – available through `PropertyMetadata.getMaxValue(String)`. The maximum value can be different for the different units.

- Enumeration of values – available through `PropertyMetadata.getEnumValues(String)`. The array of the possible values is sorted in increasing order according to the given unit.

- Resolution – available through `PropertyMetadata.getResolution(String)`. For example, if the range is [0, 100], the resolution can be 10. That's the different between two values in series. The resolution type depends on the property type. If the property is using data bean like `org.osgi.service.dal.functions.data.LevelData`, the resolution will the `BigDecimal`.

- Property access – available as a value in `PropertyMetadata.getMetadata(String)` result map. It's a bitmap of `java.lang.Integer` type and doesn't depend on the given unit. The access is available only for the function properties and it's missing for the operation arguments and result metadata. The bitmap can be any combination of:

  ○ `PropertyMetadata.PROPERTY_ACCESS_READABLE` – Marks the property as a readable. Function must provide a getter method for this property according to JavaBeans specification [3]. Function operations must not be overridden by this getter method.

  ○ `PropertyMetadata.PROPERTY_ACCESS_WRITABLE` – Marks the property as writable. Function must provide a setter method for this property according to JavaBeans specification [3]. Function operations must not be overridden by this setter method.

- ○ `PropertyMetadata.PROPERTY_ACCESS_EVENTABLE` – Marks the property as eventable. Function must not provide special methods because of this access type. `FunctionEvent` is sent on property change. Note that the event can be sent when there is no value change.

- Unit - available as a value in `PropertyMetadata.getMetadata()` result map. The value contains the property supported units. The property value type is `java.lang.String[]`. Each unit must follow those rules:

  - ○ The International System of Units must be used where it's applicable. For example, kg for kilogram and km for kilometre.

  - ○ If the unit name matches to an Unicode symbol name, the Unicode symbol must be used. For example, the degree unit matches to the Unicode degree sign (\u00B0).

  - ○ If the unit name doesn't match to an Unicode symbol, the unit symbol must be built by Unicode Basic Latin block of characters, superscript and subscript characters. For example, watt per square metre steradian is built by W/(m\u00B2 sr), where \u00B2 is Unicode superscript two.

  If those rules cannot be applied to the unit symbol, custom rules are allowed.

  A set of predefined unit symbols are available in `Units` interface.

- Description – available as a value in `PropertyMetadata.getMetadata()` result map. The property value type is `java.lang.String` and specifies an user readable description. It doesn't depend on the given unit.

- Vendor custom properties – available as a value in `PropertyMetadata.getMetadata()` result map and can depend on the given unit.

5.5.4 Function Property Event

The eventable function properties can trigger a new event on each property value touch. It doesn't require a modification of the value. For example, the motion sensor can send a few events with no property value change when motion is detected and continued to be detected. The event must implement `FunctionEvent` interface. The event properties are:

- `FunctionEvent.PROPERTY_FUNCTION_UID` – the event source function unique identifier.

- `FunctionEvent.PROPERTY_FUNCTION_PROPERTY_NAME` – the property name.

- `FunctionEvent.PROPERTY_FUNCTION_PROPERTY_VALUE` – the property value.

For example, there is function with an eventable boolean property called "state". When "state" value is changed to `false`, function implementation can post:

```
FunctionEvent {

    dal.function.UID=acme.function

    dal.function.property.name="state"

    dal.function.property.value=ACMEFuntionData(java.lang.Boolean.FALSE...)

}
```

6 Data Transfer Objects

TODO: Do we need those objects?

7 Javadoc

OSGi Javadoc

2/24/14 10:29 AM

Package Summary		Page
org.osgi.servic e.dal	Device Package Version 1.0.	*31*

Package org.osgi.service.dal

Device Package Version 1.0.

See:
> **Description**

Interface Summary		*Page*
Device	Represents the device in the OSGi service registry.	*32*
Function	Function service provides specific device operations and properties.	*48*
OperationMetadata	Contains metadata about function operation.	*61*
PropertyMetadata	Contains metadata about function property or function operation parameter.	*63*
Units	Contains the most of the International System of Units unit symbols.	*67*

Class Summary		*Page*
DevicePermission	A bundle's authority to perform specific privileged administrative operations on the devices.	*44*
FunctionData	Abstract `Function` data wrapper.	*53*
FunctionEvent	Asynchronous event, which marks a function property value modification.	*57*

Exception Summary		*Page*
DeviceException	`DeviceException` is a special `IOException`, which is thrown to indicate that there is a device operation fail.	*40*

Package org.osgi.service.dal Description

Device Package Version 1.0.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest. This package has two types of users: the consumers that use the API in this package and the providers that implement the API in this package.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.service.dal; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.service.dal; version="[1.0,1.1)"
```

Interface Device

org.osgi.service.dal

---

public interface **Device**

Represents the device in the OSGi service registry. Note that `Device` services are registered last. Before their registration, there is `Function` services registration. The reverse order is used when the services are unregistered. `Device` services are unregistered first before `Function` services.

Field Summary		*Page*
String	**DEVICE_CATEGORY** Constant for the value of the `org.osgi.service.device.Constants.DEVICE_CATEGORY` service property.	*33*
String	**SERVICE_DESCRIPTION** The service property value contains the device description.	*36*
String	**SERVICE_DRIVER** The service property value contains the device driver name.	*34*
String	**SERVICE_FIRMWARE_VENDOR** The service property value contains the device firmware vendor.	*35*
String	**SERVICE_FIRMWARE_VERSION** The service property value contains the device firmware version.	*35*
String	**SERVICE_HARDWARE_VENDOR** The service property value contains the device hardware vendor.	*35*
String	**SERVICE_HARDWARE_VERSION** The service property value contains the device hardware version.	*35*
String	**SERVICE_MODEL** The service property value contains the device model.	*36*
String	**SERVICE_NAME** The service property value contains the device name.	*34*
String	**SERVICE_REFERENCE_UIDS** The service property value contains the reference device unique identifiers.	*34*
String	**SERVICE_SERIAL_NUMBER** The service property value contains the device serial number.	*36*
String	**SERVICE_STATUS** The service property value contains the device status.	*34*
String	**SERVICE_STATUS_DETAIL** The service property value contains the device status detail.	*35*
String	**SERVICE_TYPES** The service property value contains the device types like DVD, TV etc.	*35*
String	**SERVICE_UID** The service property value contains the device unique identifier.	*34*
Integer	**STATUS_DETAIL_CONFIGURATION_NOT_APPLIED** Device status detail indicates that the device configuration is not applied.	*37*
Integer	**STATUS_DETAIL_CONNECTING** Device status detail indicates that the device is currently connecting to the network.	*37*
Integer	**STATUS_DETAIL_DEVICE_BROKEN** Device status detail indicates that the device is broken.	*37*

Integer	**STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR** Device status detail indicates that the device communication is problematic.	*38*
Integer	**STATUS_DETAIL_DEVICE_DATA_INSUFFICIENT** Device status detail indicates that the device doesn't provide enough information and cannot be determined.	*38*
Integer	**STATUS_DETAIL_DEVICE_NOT_ACCESSIBLE** Device status detail indicates that the device is not accessible and further communication is not possible.	*38*
Integer	**STATUS_DETAIL_ERROR_APPLYING_CONFIGURATION** Device status detail indicates that the device cannot be configured.	*38*
Integer	**STATUS_DETAIL_IN_DUTY_CYCLE** Device status detail indicates that the device is in duty cycle.	*38*
Integer	**STATUS_DETAIL_INITIALIZING** Device status detail indicates that the device is currently in process of initialization.	*37*
Integer	**STATUS_DETAIL_REMOVING** Device status detail indicates that the device is leaving the network.	*37*
Integer	**STATUS_NOT_CONFIGURED** Device status indicates that the device is currently not configured.	*37*
Integer	**STATUS_NOT_INITIALIZED** Device status indicates that the device is currently not initialized.	*37*
Integer	**STATUS_OFFLINE** Device status indicates that the device is currently not available for operations.	*36*
Integer	**STATUS_ONLINE** Device status indicates that the device is currently available for operations.	*36*
Integer	**STATUS_PROCESSING** Device status indicates that the device is currently busy with an operation.	*36*
Integer	**STATUS_REMOVED** Device status indicates that the device is removed from the network.	*36*

Method Summary		*Page*
Object	**getServiceProperty**(String propName) Returns the current value of the specified property.	*38*
void	**remove**() Removes this device.	*39*

Field Detail

DEVICE_CATEGORY

```
public static final String DEVICE_CATEGORY = "DAL"
```

> Constant for the value of the `org.osgi.service.device.Constants.DEVICE_CATEGORY` service property. That category is used by all device services.

> **See Also:**
>> `org.osgi.service.device.Constants.DEVICE_CATEGORY`

SERVICE_UID

```
public static final String SERVICE_UID = "dal.device.UID"
```

The service property value contains the device unique identifier. It's a mandatory property. The value type is `java.lang.String`. To simplify the unique identifier generation, the property value must follow the rule:

UID ::= driver-name ':' device-id

UID - device unique identifier

driver-name - the value of the SERVICE_DRIVER service property

device-id - device unique identifier in the scope of the driver

SERVICE_REFERENCE_UIDS

```
public static final String SERVICE_REFERENCE_UIDS = "dal.device.reference.UIDs"
```

The service property value contains the reference device unique identifiers. It's an optional property. The value type is `java.lang.String[]`. It can be used to represent different relationships between the devices. For example, the ZigBee controller can have a reference to the USB dongle.

SERVICE_DRIVER

```
public static final String SERVICE_DRIVER = "dal.device.driver"
```

The service property value contains the device driver name. For example, ZigBee, Z-Wave, Bluetooth etc. It's a mandatory property. The value type is `java.lang.String`.

SERVICE_NAME

```
public static final String SERVICE_NAME = "dal.device.name"
```

The service property value contains the device name. It's an optional property. The value type is `java.lang.String`.

SERVICE_STATUS

```
public static final String SERVICE_STATUS = "dal.device.status"
```

The service property value contains the device status. It's a mandatory property. The value type is `java.lang.Integer`. The possible values are:

- STATUS_ONLINE
- STATUS_OFFLINE
- STATUS_REMOVED
- STATUS_PROCESSING
- STATUS_NOT_INITIALIZED
- STATUS_NOT_CONFIGURED

SERVICE_STATUS_DETAIL

`public static final String` **`SERVICE_STATUS_DETAIL`** `= "dal.device.status.detail"`

The service property value contains the device status detail. It holds the reason for the current device status. It's an optional property. The value type is `java.lang.Integer`. There are two value categories:

- positive values i.e. > 0
- - Those values contain details related to the current status. Examples: STATUS_DETAIL_CONNECTING and STATUS_DETAIL_INITIALIZING.
- negative values i.e. 0
- - Those values contain errors related to the current status. Examples: STATUS_DETAIL_CONFIGURATION_NOT_APPLIED, STATUS_DETAIL_DEVICE_BROKEN and STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR.

SERVICE_HARDWARE_VENDOR

`public static final String` **`SERVICE_HARDWARE_VENDOR`** `= "dal.device.hardware.vendor"`

The service property value contains the device hardware vendor. It's an optional property. The value type is `java.lang.String`.

SERVICE_HARDWARE_VERSION

`public static final String` **`SERVICE_HARDWARE_VERSION`** `= "dal.device.hardware.version"`

The service property value contains the device hardware version. It's an optional property. The value type is `java.lang.String`.

SERVICE_FIRMWARE_VENDOR

`public static final String` **`SERVICE_FIRMWARE_VENDOR`** `= "dal.device.firmware.vendor"`

The service property value contains the device firmware vendor. It's an optional property. The value type is `java.lang.String`.

SERVICE_FIRMWARE_VERSION

`public static final String` **`SERVICE_FIRMWARE_VERSION`** `= "dal.device.firmware.version"`

The service property value contains the device firmware version. It's an optional property. The value type is `java.lang.String`.

SERVICE_TYPES

`public static final String` **`SERVICE_TYPES`** `= "dal.device.types"`

The service property value contains the device types like DVD, TV etc. It's an optional property. The value type is `java.lang.String[]`.

SERVICE_MODEL

`public static final String `**`SERVICE_MODEL`**` = "dal.device.model"`

> The service property value contains the device model. It's an optional property. The value type is `java.lang.String`.

SERVICE_SERIAL_NUMBER

`public static final String `**`SERVICE_SERIAL_NUMBER`**` = "dal.device.serial.number"`

> The service property value contains the device serial number. It's an optional property. The value type is `java.lang.String`.

SERVICE_DESCRIPTION

`public static final String `**`SERVICE_DESCRIPTION`**` = "dal.device.description"`

> The service property value contains the device description. It's an optional property. The value type is `java.lang.String`.

STATUS_REMOVED

`public static final Integer `**`STATUS_REMOVED`**

> Device status indicates that the device is removed from the network. That status must be set as the last device status and after that the device service can be unregistered from the service registry. It can be used as a value of SERVICE_STATUS service property.

STATUS_OFFLINE

`public static final Integer `**`STATUS_OFFLINE`**

> Device status indicates that the device is currently not available for operations. It can be used as a value of SERVICE_STATUS service property.

STATUS_ONLINE

`public static final Integer `**`STATUS_ONLINE`**

> Device status indicates that the device is currently available for operations. It can be used as a value of SERVICE_STATUS service property.

STATUS_PROCESSING

`public static final Integer `**`STATUS_PROCESSING`**

> Device status indicates that the device is currently busy with an operation. It can be used as a value of SERVICE_STATUS service property.

STATUS_NOT_INITIALIZED

public static final Integer **STATUS_NOT_INITIALIZED**

Device status indicates that the device is currently not initialized. Some protocols don't provide device information right after the device is connected. The device can be initialized later when it's awakened. It can be used as a value of <u>SERVICE_STATUS</u> service property.

STATUS_NOT_CONFIGURED

public static final Integer **STATUS_NOT_CONFIGURED**

Device status indicates that the device is currently not configured. The device can require additional actions to become completely connected to the network. It can be used as a value of <u>SERVICE_STATUS</u> service property.

STATUS_DETAIL_CONNECTING

public static final Integer **STATUS_DETAIL_CONNECTING**

Device status detail indicates that the device is currently connecting to the network. It can be used as a value of <u>SERVICE_STATUS_DETAIL</u> service property. The device status must be <u>STATUS_PROCESSING</u>.

STATUS_DETAIL_INITIALIZING

public static final Integer **STATUS_DETAIL_INITIALIZING**

Device status detail indicates that the device is currently in process of initialization. It can be used as a value of <u>SERVICE_STATUS_DETAIL</u> service property. The device status must be <u>STATUS_PROCESSING</u>.

STATUS_DETAIL_REMOVING

public static final Integer **STATUS_DETAIL_REMOVING**

Device status detail indicates that the device is leaving the network. It can be used as a value of <u>SERVICE_STATUS_DETAIL</u> service property. The device status must be <u>STATUS_PROCESSING</u>.

STATUS_DETAIL_CONFIGURATION_NOT_APPLIED

public static final Integer **STATUS_DETAIL_CONFIGURATION_NOT_APPLIED**

Device status detail indicates that the device configuration is not applied. It can be used as a value of <u>SERVICE_STATUS_DETAIL</u> service property. The device status must be <u>STATUS_NOT_CONFIGURED</u>.

STATUS_DETAIL_DEVICE_BROKEN

public static final Integer **STATUS_DETAIL_DEVICE_BROKEN**

Device status detail indicates that the device is broken. It can be used as a value of SERVICE_STATUS_DETAIL service property. The device status must be STATUS_OFFLINE.

STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR

`public static final Integer` **`STATUS_DETAIL_DEVICE_COMMUNICATION_ERROR`**

Device status detail indicates that the device communication is problematic. It can be used as a value of SERVICE_STATUS_DETAIL service property. The device status must be STATUS_ONLINE or STATUS_NOT_INITIALIZED.

STATUS_DETAIL_DEVICE_DATA_INSUFFICIENT

`public static final Integer` **`STATUS_DETAIL_DEVICE_DATA_INSUFFICIENT`**

Device status detail indicates that the device doesn't provide enough information and cannot be determined. It can be used as a value of SERVICE_STATUS_DETAIL service property. The device status must be STATUS_NOT_INITIALIZED.

STATUS_DETAIL_DEVICE_NOT_ACCESSIBLE

`public static final Integer` **`STATUS_DETAIL_DEVICE_NOT_ACCESSIBLE`**

Device status detail indicates that the device is not accessible and further communication is not possible. It can be used as a value of SERVICE_STATUS_DETAIL service property. The device status must be STATUS_OFFLINE.

STATUS_DETAIL_ERROR_APPLYING_CONFIGURATION

`public static final Integer` **`STATUS_DETAIL_ERROR_APPLYING_CONFIGURATION`**

Device status detail indicates that the device cannot be configured. It can be used as a value of SERVICE_STATUS_DETAIL service property. The device status must be STATUS_NOT_CONFIGURED.

STATUS_DETAIL_IN_DUTY_CYCLE

`public static final Integer` **`STATUS_DETAIL_IN_DUTY_CYCLE`**

Device status detail indicates that the device is in duty cycle. It can be used as a value of SERVICE_STATUS_DETAIL service property. The device status must be STATUS_OFFLINE.

Method Detail

getServiceProperty

`Object` **`getServiceProperty`**`(String propName)`

Returns the current value of the specified property. The method will return the same value as `org.osgi.framework.ServiceReference.getProperty(String)` for the service reference of this device.

This method must continue to return property values after the device service has been unregistered.

Parameters:
> `propName` - The property name.

Returns:
> The property value or `null` if the property name cannot be mapped to a value.

---

remove

```
void remove()
     throws DeviceException,
          UnsupportedOperationException,
          SecurityException,
          IllegalStateException
```

Removes this device. The method must synchronously remove the device from the device network.

Throws:
> DeviceException - If an operation error is available.
>
> UnsupportedOperationException - If the operation is not supported over this device.
>
> SecurityException - If the caller does not have the appropriate FunctionalDevicePermission[this device, DevicePermission.ACTION_REMOVE] and the Java Runtime Environment supports permissions.
>
> IllegalStateException - If this device service object has already been unregistered.

Class DeviceException

org.osgi.service.dal

```
java.lang.Object
  └─ java.lang.Throwable
       └─ java.lang.Exception
            └─ java.io.IOException
                 └─ org.osgi.service.dal.DeviceException
```

All Implemented Interfaces:
> Serializable

---

public class **DeviceException**
extends IOException

`DeviceException` is a special `IOException`, which is thrown to indicate that there is a device operation fail. The error reason can be located with getCode() method. The cause is available with getCause().

Field Summary		*Page*
static int	**CODE_COMMUNICATION_ERROR** An exception code indicates that there is an error in the communication.	*41*
static int	**CODE_NO_DATA** An exception code indicates that the requested value is currently not available.	*41*
static int	**CODE_NOT_INITIALIZED** An exception code indicates that the device is not initialized.	*41*
static int	**CODE_TIMEOUT** An exception code indicates that there is expired timeout without any processing.	*41*
static int	**CODE_UNKNOWN** An exception code indicates that the error is unknown.	*41*

Constructor Summary	*Page*
DeviceException() Construct a new device exception with `null` message.	*41*
DeviceException(String message) Constructs a new device exception with the given message.	*42*
DeviceException(String message, Throwable cause) Constructs a new device exception with the given message and cause.	*42*
DeviceException(String message, Throwable cause, int code) Constructs a new device exception with the given message, cause and code.	*42*

Method Summary		*Page*
Throwable	**getCause**() Returns the cause for this exception or `null` if the cause is missing.	*42*
int	**getCode**() Returns the exception error code.	*42*
void	**printStackTrace**() Prints the exception stack trace to the standard error stream.	*43*

---

void	**printStackTrace**(PrintStream s) Prints the exception stack trace to the given stream.	*43*
void	**printStackTrace**(PrintWriter s) Prints the exception stack trace to the given writer.	*43*

Field Detail

CODE_UNKNOWN

public static final int **CODE_UNKNOWN** = 1

An exception code indicates that the error is unknown.

CODE_COMMUNICATION_ERROR

public static final int **CODE_COMMUNICATION_ERROR** = 2

An exception code indicates that there is an error in the communication.

CODE_TIMEOUT

public static final int **CODE_TIMEOUT** = 3

An exception code indicates that there is expired timeout without any processing.

CODE_NOT_INITIALIZED

public static final int **CODE_NOT_INITIALIZED** = 4

An exception code indicates that the device is not initialized. The device status is Device.STATUS_NOT_INITIALIZED or Device.STATUS_PROCESSING.

CODE_NO_DATA

public static final int **CODE_NO_DATA** = 5

An exception code indicates that the requested value is currently not available.

Constructor Detail

DeviceException

public **DeviceException**()

Construct a new device exception with null message. The cause is not initialized and the exception code is set to CODE_UNKNOWN.

DeviceException

public **DeviceException**(String message)

> Constructs a new device exception with the given message. The cause is not initialized and the exception code is set to CODE_UNKNOWN.
>
> **Parameters:**
> > `message` - The excpetion message.

DeviceException

public **DeviceException**(String message,
                    Throwable cause)

> Constructs a new device exception with the given message and cause. The exception code is set to CODE_UNKNOWN.
>
> **Parameters:**
> > `message` - The exception message.
> > `cause` - The exception cause.

DeviceException

public **DeviceException**(String message,
                    Throwable cause,
                    int code)

> Constructs a new device exception with the given message, cause and code.
>
> **Parameters:**
> > `message` - The exception message.
> > `cause` - The exception cause.
> > `code` - The exception code.

Method Detail

getCode

public int **getCode**()

> Returns the exception error code. It indicates the reason for this exception.
>
> **Returns:**
> > An exception code.

getCause

public Throwable **getCause**()

Returns the cause for this exception or `null` if the cause is missing. The cause can be protocol specific exception with an appropriate message and error code.

Overrides:
getCause in class Throwable
Returns:
An throwable cause.

printStackTrace

public void **printStackTrace**()

Prints the exception stack trace to the standard error stream.

Overrides:
printStackTrace in class Throwable
See Also:
Throwable.printStackTrace()

printStackTrace

public void **printStackTrace**(PrintStream s)

Prints the exception stack trace to the given stream.

Overrides:
printStackTrace in class Throwable
Parameters:
s - The stream used for the output.
See Also:
Throwable.printStackTrace(java.io.PrintStream)

printStackTrace

public void **printStackTrace**(PrintWriter s)

Prints the exception stack trace to the given writer.

Overrides:
printStackTrace in class Throwable
Parameters:
s - The writer used for the output.
See Also:
Throwable.printStackTrace(java.io.PrintWriter)

Class DevicePermission

org.osgi.service.dal

```
java.lang.Object
  └─java.security.Permission
      └─java.security.BasicPermission
          └─org.osgi.service.dal.DevicePermission
```

All Implemented Interfaces:
> Guard, Serializable

```
final public class DevicePermission
extends BasicPermission
```

A bundle's authority to perform specific privileged administrative operations on the devices. The actions for this permission are:

Action	Method
ACTION_REMOVE	Device.remove()

The name of the permission is a filter based. See OSGi Core Specification, Filter Based Permissions. The filter gives an access to all device service properties. The service property names are case insensitive. The filter attribute names are processed in a case insensitive manner.

Field Summary	*Page*
static String **ACTION_REMOVE** A permission action to remove the device.	*45*

Constructor Summary	*Page*
DevicePermission(String filter, String action) Creates a new `FunctionalDevicePermission` with the given filter and actions.	*45*
DevicePermission(Device device, String action) Creates a new `FunctionalDevicePermission` with the given device and actions.	*45*

Method Summary	*Page*

boolean **equals**(Object obj)
       Two `FunctionalDevicePermission` instances are equal if:
      • represents the same filter and actions
      • represents the same device and actions
	45

String **getActions**() Returns the canonical string representation of ACTION_REMOVE action.	*46*
int **hashCode**() Returns the hash code value for this object.	*46*
boolean **implies**(Permission p) Determines if the specified permission is implied by this object.	*46*
Permission Collection **newPermissionCollection**() Returns a new `PermissionCollection` suitable for storing `FunctionalDevicePermission` instances.	*47*

Field Detail

ACTION_REMOVE

```
public static final String ACTION_REMOVE = "remove"
```

A permission action to remove the device.

Constructor Detail

DevicePermission

```
public DevicePermission(String filter,
                        String action)
```

Creates a new `FunctionalDevicePermission` with the given filter and actions. The constructor must only be used to create a permission that is going to be checked.

An filter example: (dal.device.hardware.vendor=acme)

An action list example: property, remove

Parameters:
> `filter` - A filter expression that can use any device service property. The filter attribute names are processed in a case insensitive manner. A special value of "*" can be used to match all devices.
> `action` - ACTION_REMOVE action.

Throws:
> `IllegalArgumentException` - If the filter syntax is not correct or invalid actions are specified.

DevicePermission

```
public DevicePermission(Device device,
                        String action)
```

Creates a new `FunctionalDevicePermission` with the given device and actions. The permission must be used for the security checks like:

`securityManager.checkPermission(new FunctionalDevicePermission(this, "remove")); .` The permissions constructed by this constructor must not be added to the `FunctionalDevicePermission` permission collections.

Parameters:
> `device` - The permission device.
> `action` - ACTION_REMOVE action.

Method Detail

equals

```
public boolean equals(Object obj)
```

Two `FunctionalDevicePermission` instances are equal if:

- represents the same filter and actions
- represents the same device and actions

> **Overrides:**
> > `equals` in class `BasicPermission`
>
> **Parameters:**
> > `obj` - The object being compared for equality with this object.
>
> **Returns:**
> > `true` if two permissions are equal, `false` otherwise.

---

hashCode

`public int hashCode()`

> Returns the hash code value for this object.

> **Overrides:**
> > `hashCode` in class `BasicPermission`
>
> **Returns:**
> > Hash code value for this object.

---

getActions

`public String getActions()`

> Returns the canonical string representation of <u>ACTION_REMOVE</u> action.

> **Overrides:**
> > `getActions` in class `BasicPermission`
>
> **Returns:**
> > The canonical string representation of the actions.

---

implies

`public boolean implies(Permission p)`

> Determines if the specified permission is implied by this object. The method will throw an exception if the specified permission was not constructed by <u>DevicePermission(Device, String)</u>. Returns `true` if the specified permission is a `FunctionalDevicePermission` and this permission filter matches the specified permission device properties.

> **Overrides:**
> > `implies` in class `BasicPermission`
>
> **Parameters:**
> > `p` - The permission to be implied. It must be constructed by <u>DevicePermission(Device, String)</u>.
>
> **Returns:**
> > `true` if the specified permission is implied by this permission, `false` otherwise.
>
> **Throws:**
> > `IllegalArgumentException` - If the specified permission is not constructed by <u>DevicePermission(Device, String)</u>.

---

newPermissionCollection

`public PermissionCollection` **`newPermissionCollection`**`()`

> Returns a new `PermissionCollection` suitable for storing `FunctionalDevicePermission` instances.

> **Overrides:**
> > `newPermissionCollection` in class `BasicPermission`
> **Returns:**
> > A new `PermissionCollection` instance.

newPermissionCollection

`public PermissionCollection` **`newPermissionCollection`**`()`

Interface Function

org.osgi.service.dal

```
public interface Function
```

Function service provides specific device operations and properties. Each function service must implement this interface. In additional to this interface, the implementation can provide own:

- properties;
- operations.

The function service can be registered in the service registry with those service properties:

- SERVICE_UID - mandatory service property. The property value contains the function unique identifier.
- SERVICE_DEVICE_UID - optional service property. The property value is the Functional Device identifiers. The function belongs to those devices.
- SERVICE_REFERENCE_UIDS - optional service property. The property value contains the reference function unique identifiers.
- SERVICE_TYPE - mandatory service property. The property value is the function type.
- SERVICE_VERSION - optional service property. The property value contains the function version.
- SERVICE_DESCRIPTION - optional service property. The property value is the function description.
- SERVICE_OPERATION_NAMES - optional service property. The property value is the function operation names.
- SERVICE_PROPERTY_NAMES - optional service property. The property value is the function property names.

The Function services are registered before the Device services. It's possible that SERVICE_DEVICE_UID point to missing services at the moment of the registration. The reverse order is used when the services are unregistered. Function services are unregistered last after Device services.

Function service must be registered only under concrete function class. It's not allowed to register function service under more than one class. For example, those registrations are not allowed:

- context.registerService(ManagedService.class.getName(), this, regProps); - ManagedService interface is not a function interface.
- context.registerService(Function.class.getName(), this, regProps); - Function interface is not concrete function interface.
- context.registerService(new String[] {BooleanControl.class.getName(), BooleanControl.class.getName()}, this, regProps); - more than one function is used.

That one is a valid registration: context.registerService(Meter.class.getName(), this, regProps);. Meter is concrete function interface.

That rule helps to the applications to find the supported function class and to identify the metadata. Otherwise the function services can be accesses, but it's not clear which are the function classes and metadata.

The function properties must be integrated according to these rules:

- Getter methods must be available for all properties with PropertyMetadata.PROPERTY_ACCESS_READABLE access.
- Getter method must return a subclass of FunctionData.
- Setter methods must be available for all properties with PropertyMetadata.PROPERTY_ACCESS_WRITABLE access.
- Setter method must use FunctionData wrapped type. For example, there is MyFunctionData with timestamp, unit and BigDecimal value. The setter must accept as an argument the value of type BigDecimal.

- It's possible to have a second setter method, which accepts the value as a first argument and the unit as a second argument.
- No methods are required for properties with `PropertyMetadata.PROPERTY_ACCESS_EVENTABLE` access.

The accessor method names must be defined according JavaBeans specification.

The function operations are java methods, which cannot override the property accessor methods. They can have zero or more parameters and zero or one return value.

Operation arguments and function properties are restricted by the same set of rules. The data type can be one of the following types:

- Java primitive type or corresponding reference type.
- java.lang.String.
- Beans, but the beans properties must use those rules. Java Beans are defined in JavaBeans specification.
- java.util.Maps. The keys can be any reference type of Java primitive types or java.lang.String. The values must use those rules.
- Arrays of defined types.

The properties metadata is accessible with `getPropertyMetadata(String)`. The operations metadata is accessible with `getOperationMetadata(String)`. In order to provide common behavior, all functions must follow a set of common rules related to the implementation of their setters, getters, operations and events:

- The setter method must be executed synchronously. If the underlying protocol can return response to the setter call, it must be awaited. It simplifies the property value modifications and doesn't require asynchronous callback.
- The operation method must be executed synchronously. If the underlying protocol can return an operation confirmation or response, they must be awaited. It simplifies the operation execution and doesn't require asynchronous callback.
- The getter must return the last know cached property value. The device implementation is responsible to keep that value up to date. It'll speed up the applications when the function property values are collected. The same cached value can be shared between a few requests instead of a few calls to the real device.
- If a given function operation, getter or setter is not supported, java.lang.UnsupportedOperationException must be thrown. It indicates that function is partially supported.
- The function operations, getters and setters must not override java.lang.Object and this interface methods.

Field Summary		*Page*
`String`	**SERVICE_DESCRIPTION** The service property value contains the function description.	*51*
`String`	**SERVICE_DEVICE_UID** The service property value contains the device unique identifier.	*51*
`String`	**SERVICE_OPERATION_NAMES** The service property value contains the function operation names.	*51*
`String`	**SERVICE_PROPERTY_NAMES** The service property value contains the function property names.	*51*
`String`	**SERVICE_REFERENCE_UIDS** The service property value contains the reference function unique identifiers.	*51*
`String`	**SERVICE_TYPE** The service property value contains the function type.	*50*

String	**SERVICE_UID** The service property value contains the function unique identifier.	*50*
String	**SERVICE_VERSION** The service property value contains the function version.	*50*

Method Summary		*Page*
OperationM etadata	**getOperationMetadata**(String operationName) Provides metadata about the function operation.	*52*
PropertyMe tadata	**getPropertyMetadata**(String propertyName) Provides metadata about the function property specified with the name argument.	*51*
Object	**getServiceProperty**(String propName) Returns the current value of the specified property.	*52*

Field Detail

SERVICE_UID

public static final String **SERVICE_UID** = "dal.function.UID"

The service property value contains the function unique identifier. It's a mandatory property. The value type is `java.lang.String`. To simplify the unique identifier generation, the property value must follow the rule:

function UID ::= device-id ':' function-id

function UID - function unique identifier

device-id - the value of the Device.SERVICE_UID Device service property

function-id - function identifier in the scope of the device

If the function is not bound to a device, the function unique identifier can be device independent.

SERVICE_TYPE

public static final String **SERVICE_TYPE** = "dal.function.type"

The service property value contains the function type. It's an optional property. For example, the sensor function can have different types like temperature or pressure etc. The value type is `java.lang.String`.

Organizations that want to use function types that do not clash with OSGi Alliance defined types should prefix their types in own namespace.

The type does'nt mandate specific function interface. It can be used with different functions.

SERVICE_VERSION

public static final String **SERVICE_VERSION** = "dal.function.version"

The service property value contains the function version. That version can point to specific implementation version and vary in the different vendor implementations. It's an optional property. The value type is `java.lang.String`.

SERVICE_DEVICE_UID

```
public static final String SERVICE_DEVICE_UID = "dal.function.device.UID"
```

> The service property value contains the device unique identifier. The function belongs to this device. It's an optional property. The value type is `java.lang.String`.

SERVICE_REFERENCE_UIDS

```
public static final String SERVICE_REFERENCE_UIDS = "dal.function.reference.UIDs"
```

> The service property value contains the reference function unique identifiers. It's an optional property. The value type is `java.lang.String[]`. It can be used to represent different relationships between the functions.

SERVICE_DESCRIPTION

```
public static final String SERVICE_DESCRIPTION = "dal.function.description"
```

> The service property value contains the function description. It's an optional property. The value type is `java.lang.String`.

SERVICE_OPERATION_NAMES

```
public static final String SERVICE_OPERATION_NAMES = "dal.function.operation.names"
```

> The service property value contains the function operation names. It's an optional property. The value type is `java.lang.String[]`. It's not possible to exist two or more function operations with the same name i.e. the operation overloading is not allowed.

SERVICE_PROPERTY_NAMES

```
public static final String SERVICE_PROPERTY_NAMES = "dal.function.property.names"
```

> The service property value contains the function property names. It's an optional property. The value type is `java.lang.String[]`. It's not possible to exist two or more function properties with the same name.

Method Detail

getPropertyMetadata

```
PropertyMetadata getPropertyMetadata(String propertyName)
                            throws IllegalArgumentException
```

> Provides metadata about the function property specified with the name argument.
>
> This method must continue to return the property metadata after the function service has been unregistered.

Parameters:

   propertyName - The function property name, which metadata is requested.

Returns:

   The property metadata for the given property name. null if the property metadata is not supported.

Throws:

   IllegalArgumentException - If the function property with the specified name is not supported.

getOperationMetadata

```
OperationMetadata getOperationMetadata(String operationName)
                                  throws IllegalArgumentException
```

Provides metadata about the function operation.

This method must continue to return the operation metadata after the function service has been unregistered.

Parameters:

   operationName - The function operation name, which metadata is requested.

Returns:

   The operation metadata for the given operation name. null if the operation metadata is not supported.

Throws:

   IllegalArgumentException - If the function operation with the specified name is not supported.

getServiceProperty

```
Object getServiceProperty(String propName)
```

Returns the current value of the specified property. The method will return the same value as org.osgi.framework.ServiceReference.getProperty(String) for the service reference of this function.

This method must continue to return property values after the device function service has been unregistered.

Parameters:

   propName - The property name.

Returns:

   The property value or null if the property name cannot be mapped to a value.

Class FunctionData

org.osgi.service.dal

```
java.lang.Object
    └─org.osgi.service.dal.FunctionData
```

All Implemented Interfaces:

      Comparable

```
abstract public class FunctionData
extends Object
implements Comparable
```

Abstract `Function` data wrapper. A subclass must be used for an access to the property values by all functions. It takes care about the timestamp and additional metadata. The subclasses are responsible to provide concrete value and unit if required.

The subclass is responsible to provide correct implementation of `Comparable.compareTo(Object)` method.

Field Summary		*Page*
static String	**FIELD_METADATA** Represents the metadata field name.	*54*
static String	**FIELD_TIMESTAMP** Represents the timestamp field name.	*54*
static String	**META_INFO_DESCRIPTION** Metadata key, which value represents the data description.	*54*
Map	**metadata** Contains `FunctionData` metadata.	*54*
long	**timestamp** Contains `FunctionData` timestamp.	*54*

Constructor Summary	*Page*
FunctionData(Map fields) Constructs new `FunctionData` instance with the specified field values.	*54*
FunctionData(long timestamp, Map metadata) Constructs new `FunctionData` instance with the specified arguments.	*55*

Method Summary		*Page*
boolean	**equals**(Object other) Two `FunctionData` instances are equal if their metadata and timestamp are equivalent.	*55*
Map	**getMetadata**() Returns `FunctionData` metadata.	*55*
long	**getTimestamp**() Returns `FunctionData` timestamp.	*55*
int	**hashCode**() Returns the hash code of this `FunctionData`.	*56*

Field Detail

FIELD_TIMESTAMP

```
public static final String FIELD_TIMESTAMP = "timestamp"
```

> Represents the timestamp field name. The field value is available with <u>timestamp</u> and <u>getTimestamp()</u>. The field type is `long`. The constant can be used as a key to <u>FunctionData(Map)</u>.

FIELD_METADATA

```
public static final String FIELD_METADATA = "metadata"
```

> Represents the metadata field name. The field value is available with <u>metadata</u> and <u>getMetadata()</u>. The field type is `Map`. The constant can be used as a key to <u>FunctionData(Map)</u>.

META_INFO_DESCRIPTION

```
public static final String META_INFO_DESCRIPTION = "description"
```

> Metadata key, which value represents the data description. The property value type is `java.lang.String`.

timestamp

```
public final long timestamp
```

> Contains `FunctionData` timestamp. The timestamp is the difference between the value collecting time and midnight, January 1, 1970 UTC. It's measured in milliseconds. The device driver is responsible to generate that value when the value is received from the device. `Long.MIN_VALUE` value means no timestamp.

metadata

```
public final Map metadata
```

> Contains `FunctionData` metadata. It's dynamic metadata related only to this specific value. Possible keys:
>
> - <u>META_INFO_DESCRIPTION</u>
> - custom key

Constructor Detail

FunctionData

```
public FunctionData(Map fields)
```

> Constructs new `FunctionData` instance with the specified field values. The map keys must match to the field names. The map values will be assigned to the appropriate class fields. For example, the maps can be: {"timestamp"=Long(1384440775495)}. That map will initialize the <u>FIELD_TIMESTAMP</u> field with 1384440775495. If timestamp is missing, `Long.MIN_VALUE` is used.

[FIELD_TIMESTAMP](#) field value type must be `Long`. [FIELD_METADATA](#) field value type must be `Map`.

> **Parameters:**
> > `fields` - Contains the new `FunctionData` instance field values.
>
> **Throws:**
> > `ClassCastException` - If the field value types are not expected.
> > `NullPointerException` - If the fields map is `null`.

FunctionData

```
public FunctionData(long timestamp,
                    Map metadata)
```

> Constructs new `FunctionData` instance with the specified arguments.
>
> **Parameters:**
> > `timestamp` - The data timestamp.
> > `metadata` - The data metadata.

Method Detail

getTimestamp

```
public long getTimestamp()
```

> Returns `FunctionData` timestamp. The timestamp is the difference between the value collecting time and midnight, January 1, 1970 UTC. It's measured in milliseconds. The device driver is responsible to generate that value when the value is received from the device. `Long.MIN_VALUE` value means no timestamp.
>
> **Returns:**
> > `FunctionData` timestamp.

---

getMetadata

```
public Map getMetadata()
```

> Returns `FunctionData` metadata. It's dynamic metadata related only to this specific value. Possible keys:
>
> - [META_INFO_DESCRIPTION](#)
> - custom key
>
> **Returns:**
> > `FunctionData` metadata or `null` is there is no metadata.

---

equals

```
public boolean equals(Object other)
```

> Two `FunctionData` instances are equal if their metadata and timestamp are equivalent.

Overrides:
      `equals` in class `Object`
Parameters:
      `other` - The other instance to compare. It must be of `FunctionData` type.
Returns:
      `true` if this instance and argument have equivalent metadata and timestamp, `false` otherwise.
See Also:
      `Object.equals(java.lang.Object)`

---

hashCode

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

Returns the hash code of this `FunctionData`.

Overrides:
      `hashCode` in class `Object`
Returns:
      `FunctionData` hash code.
See Also:
      `Object.hashCode()`

Class FunctionEvent

org.osgi.service.dal

```
java.lang.Object
  └─org.osgi.service.event.Event
       └─org.osgi.service.dal.FunctionEvent
```

```
final public class FunctionEvent
extends org.osgi.service.event.Event
```

Asynchronous event, which marks a function property value modification. The event can be triggered when there is a new property value, but it's possible to have events in series with no value change. The event properties must contain:

- PROPERTY_FUNCTION_UID - the event source function unique identifier.
- PROPERTY_FUNCTION_PROPERTY_NAME - the property name.
- PROPERTY_FUNCTION_PROPERTY_VALUE - the property value. The property value type must be a subclass of FunctionData.

Field Summary		*Page*
static String	**EVENT_CLASS** Represents the event class.	*58*
static String	**EVENT_PACKAGE** Represents the event package.	*58*
static String	**PROPERTY_FUNCTION_PROPERTY_NAME** Represents an event property key for the function property name.	*58*
static String	**PROPERTY_FUNCTION_PROPERTY_VALUE** Represents an event property key for the function property value.	*58*
static String	**PROPERTY_FUNCTION_UID** Represents an event property key for function UID.	*58*
static String	**TOPIC_PROPERTY_CHANGED** Represents the event topic for the function property changed.	*58*

Constructor Summary	*Page*
FunctionEvent(String topic, String funtionUID, String propName, FunctionData propValue) Constructs a new event with the specified topic, function UID, property name and property value.	*59*
FunctionEvent(String topic, Dictionary properties) Constructs a new event with the specified topic and properties.	*59*
FunctionEvent(String topic, Map properties) Constructs a new event with the specified topic and properties.	*59*

Method Summary		*Page*
String	**getFunctionPropertyName**() Returns the property name.	*60*
FunctionData	**getFunctionPropertyValue**() Returns the property value.	*60*
String	**getFunctionUID**() Returns the property value change source function identifier.	*59*

Methods inherited from class org.osgi.service.event.Event
equals, getProperty, getPropertyNames, getTopic, hashCode, matches, toString

Field Detail

EVENT_PACKAGE

public static final String **EVENT_PACKAGE** = "org/osgi/service/dal/"

> Represents the event package. That constant can be useful for the event handlers depending on the event filters.

EVENT_CLASS

public static final String **EVENT_CLASS** = "org/osgi/service/dal/FunctionEvent/"

> Represents the event class. That constant can be useful for the event handlers depending on the event filters.

TOPIC_PROPERTY_CHANGED

public static final String **TOPIC_PROPERTY_CHANGED** = "org/osgi/service/dal/FunctionEvent/PROPERTY_CHANGED"

> Represents the event topic for the function property changed.

PROPERTY_FUNCTION_UID

public static final String **PROPERTY_FUNCTION_UID** = "dal.function.UID"

> Represents an event property key for function UID. The property value type is java.lang.String. The value represents the property value change source function identifier.

PROPERTY_FUNCTION_PROPERTY_NAME

public static final String **PROPERTY_FUNCTION_PROPERTY_NAME** = "dal.function.property.name"

> Represents an event property key for the function property name. The property value type is java.lang.String. The value represents the property name.

PROPERTY_FUNCTION_PROPERTY_VALUE

public static final String **PROPERTY_FUNCTION_PROPERTY_VALUE** = "dal.function.property.value"

> Represents an event property key for the function property value. The property value type is a subclass of FunctionData. The value represents the property value.

Constructor Detail

FunctionEvent

```
public FunctionEvent(String topic,
                     Dictionary properties)
```

Constructs a new event with the specified topic and properties.

Parameters:
        `topic` - The event topic.
        `properties` - The event properties.

FunctionEvent

```
public FunctionEvent(String topic,
                     Map properties)
```

Constructs a new event with the specified topic and properties.

Parameters:
        `topic` - The event topic.
        `properties` - The event properties.

FunctionEvent

```
public FunctionEvent(String topic,
                     String funtionUID,
                     String propName,
                     FunctionData propValue)
```

Constructs a new event with the specified topic, function UID, property name and property value.

Parameters:
        `topic` - The event topic.
        `funtionUID` - The event source function UID.
        `propName` - The event source property name.
        `propValue` - The event source property value.

Method Detail

getFunctionUID

```
public String getFunctionUID()
```

Returns the property value change source function identifier. The value is same as the value of PROPERTY_FUNCTION_UID property.

Returns:
        The property value change source function.

---

getFunctionPropertyName

```
public String getFunctionPropertyName()
```

>   Returns the property name. The value is same as the value of <u>PROPERTY_FUNCTION_PROPERTY_NAME</u>.

>   **Returns:**
>   >   The property name.

getFunctionPropertyValue

```
public FunctionData getFunctionPropertyValue()
```

>   Returns the property value. The value is same as the value of <u>PROPERTY_FUNCTION_PROPERTY_VALUE</u>.

>   **Returns:**
>   >   The property value.

Interface OperationMetadata

org.osgi.service.dal

public interface **OperationMetadata**

Contains metadata about function operation.

See Also:

>Function, PropertyMetadata

Field Summary		Page
String	**META_INFO_DESCRIPTION** Metadata key, which value represents the operation description.	*61*

Method Summary		Page
Map	**getMetadata**() Returns metadata about the function operation.	*61*
PropertyMetadata[]	**getParametersMetadata**() Returns metadata about the operation parameters or `null` if no such medatadata is available.	*62*
PropertyMetadata	**getReturnValueMetadata**() Returns metadata about the operation return value or `null` if no such metadata is available.	*62*

Field Detail

META_INFO_DESCRIPTION

public static final String **META_INFO_DESCRIPTION** = "description"

>Metadata key, which value represents the operation description. The property value type is `java.lang.String`.

Method Detail

getMetadata

Map **getMetadata**()

>Returns metadata about the function operation. The keys of the `java.util.Map` result must be of `java.lang.String` type. Possible keys:

>- META_INFO_DESCRIPTION
>- custom key

>**Returns:**
>>The operation metadata or `null` if no such metadata is available.

getReturnValueMetadata

<u>PropertyMetadata</u> **getReturnValueMetadata**()

       Returns metadata about the operation return value or `null` if no such metadata is available.

       Returns:

              Operation return value metadata.

getParametersMetadata

<u>PropertyMetadata</u>[] **getParametersMetadata**()

       Returns metadata about the operation parameters or `null` if no such medatadata is available.

       Returns:

              Operation parameters medata.

Interface PropertyMetadata

org.osgi.service.dal

```
public interface PropertyMetadata
```

Contains metadata about function property or function operation parameter. The access to the function properties is a bitmap value of PROPERTY_ACCESS metadata key. Function properties can be accessed in three ways. Any combinations between them are possible:

- PROPERTY_ACCESS_READABLE - available for all properties, which can be read. Function must provide a getter method for an access to the property value.
- PROPERTY_ACCESS_WRITABLE - available for all properties, which can be modified. Function must provide a setter method for a modification of the property value.
- PROPERTY_ACCESS_EVENTABLE - available for all properties, which can report the property value. FunctionEvents are sent on property change.

See Also:
> Function, PropertyMetadata

Field Summary		*Page*
String	**DESCRIPTION** Metadata key, which value represents the property description.	*64*
String	**PROPERTY_ACCESS** Metadata key, which value represents the access to the function property.	*64*
int	**PROPERTY_ACCESS_EVENTABLE** Marks the eventable function properties.	*64*
int	**PROPERTY_ACCESS_READABLE** Marks the readable function properties.	*64*
int	**PROPERTY_ACCESS_WRITABLE** Marks the writable function properties.	*64*
String	**UNITS** Metadata key, which value represents the property supported units.	*65*

Method Summary		*Page*
FunctionData[]	**getEnumValues**(String unit) Returns the property possible values according to the specified unit.	*66*
FunctionData	**getMaxValue**(String unit) Returns the property maximum value according to the specified unit.	*66*
Map	**getMetadata**(String unit) Returns metadata about the function property or operation parameter.	*65*
FunctionData	**getMinValue**(String unit) Returns the property minimum value according to the specified unit.	*66*
Object	**getResolution**(String unit) Returns the resolution value of specific range.	*65*

Field Detail

PROPERTY_ACCESS_READABLE

`public static final int` **`PROPERTY_ACCESS_READABLE`** `= 1`

> Marks the readable function properties. The flag can be used as a part of bitmap value of PROPERTY_ACCESS. The readable access mandates function to provide a property getter method.
>
> **See Also:**
> > Function

PROPERTY_ACCESS_WRITABLE

`public static final int` **`PROPERTY_ACCESS_WRITABLE`** `= 2`

> Marks the writable function properties. The flag can be used as a part of bitmap value of PROPERTY_ACCESS. The writable access mandates function to provide a property setter methods.
>
> **See Also:**
> > Function

PROPERTY_ACCESS_EVENTABLE

`public static final int` **`PROPERTY_ACCESS_EVENTABLE`** `= 4`

> Marks the eventable function properties. The flag can be used as a part of bitmap value of PROPERTY_ACCESS.
>
> **See Also:**
> > Function

PROPERTY_ACCESS

`public static final String` **`PROPERTY_ACCESS`** `= "property.access"`

> Metadata key, which value represents the access to the function property. The property value is a bitmap of `Integer` type. The bitmap can be any combination of:
>
> - PROPERTY_ACCESS_READABLE
> - PROPERTY_ACCESS_WRITABLE
> - PROPERTY_ACCESS_EVENTABLE
>
> For example, value Integer(3) means that the property is readable and writable, but not eventable.
>
> The property access is available only for function properties and it's missing for the operation parameters.

DESCRIPTION

`public static final String` **`DESCRIPTION`** `= "description"`

Metadata key, which value represents the property description. The property value type is `java.lang.String`.

UNITS

```
public static final String UNITS = "units"
```

Metadata key, which value represents the property supported units. The property value type is `java.lang.String[]`. Each unit must follow those rules:

- The International System of Units must be used where it's applicable. For example, kg for kilogram and km for kilometre.
- If the unit name matches to an Unicode symbol name, the Unicode symbol must be used. For example, the degree unit matches to the Unicode degree sign (°).
- If the unit name doesn't match to an Unicode symbol, the unit symbol must be built by Unicode Basic Latin block of characters, superscript and subscript characters. For example, watt per square metre steradian is built by W/(m² sr), where ² is Unicode superscript two.

If those rules cannot be applied to the unit symbol, custom rules are allowed. A set of predefined unit symbols are available in <u>Units</u> interface.

Method Detail

getMetadata

```
Map getMetadata(String unit)
```

Returns metadata about the function property or operation parameter. The keys of the `java.util.Map` result must be of `java.lang.String` type. Possible keys:

- <u>DESCRIPTION</u> - doesn't depend on the given unit.
- <u>PROPERTY_ACCESS</u> - available only for function property and missing for function operation parameters. It doesn't depend on the given unit.
- <u>UNITS</u> - doesn't depend on the given unit.
- custom key - can depend on the unit.

Parameters:
> `unit` - The unit to align the metadata if it's applicable. It can be null, which means that the default unit will be used.

Returns:
> The property metadata or `null` if no such metadata is available.

---

getResolution

```
Object getResolution(String unit)
          throws IllegalArgumentException
```

Returns the resolution value of specific range. For example, if the range is [0, 100], the resolution can be 10. That's the different between two values in series. The resolution type depends on the property type. If the property is using data bean like `LevelData`, the resolution will the `BigDecimal`.

Parameters:
> `unit` - The unit to align the resolution, can be `null`.

Returns:

> The resolution according to the specified unit or `null` if no resolution is supported.

Throws:

> `IllegalArgumentException` - If the unit is not supported.

---

getEnumValues

`FunctionData[]` **getEnumValues**(String unit)
                    throws IllegalArgumentException

Returns the property possible values according to the specified unit. If the unit is `null`, the values set is aligned to the default unit. If there is no such set of supported values, `null` is returned. The values must be sorted in increasing order.

Parameters:

> `unit` - The unit to align the supported values, can be `null`.

Returns:

> The supported values according to the specified unit or `null` if no such values are supported. The values must be sorted in increasing order.

Throws:

> `IllegalArgumentException` - If the unit is not supported.

---

getMinValue

`FunctionData` **getMinValue**(String unit)
                throws IllegalArgumentException

Returns the property minimum value according to the specified unit. If the unit is `null`, the minimum value is aligned to the default unit. If there is no minimum value, `null` is returned.

Parameters:

> `unit` - The unit to align the minimum value, can be `null` .

Returns:

> The minimum value according to the specified unit or `null` if no minimum value is supported.

Throws:

> `IllegalArgumentException` - If the unit is not supported.

---

getMaxValue

`FunctionData` **getMaxValue**(String unit)
                throws IllegalArgumentException

Returns the property maximum value according to the specified unit. If the unit is `null`, the maximum value is aligned to the default unit. If there is no maximum value, `null` is returned.

Parameters:

> `unit` - The unit to align the maximum value, can be `null` .

Returns:

> The maximum value according to the specified unit or `null` if no maximum value is supported.

Throws:

> `IllegalArgumentException` - If the unit is not supported.

Interface Units

org.osgi.service.dal

---

public interface **Units**

Contains the most of the International System of Units unit symbols. The constant name represents the unit name. The constant value represents the unit symbol as it's defined in **PropertyMetadata.UNITS**.

Field Summary		*Page*
String	**AMPERE** Unit of electric current defined by the International System of Units (SI).	*72*
String	**AMPERE_PER_METRE** Unit of magnetic field strength.	*74*
String	**AMPERE_PER_SQUARE_METRE** Unit of current density.	*74*
String	**ANGSTROM** Unit of length.	*82*
String	**BAR** Unit of pressure.	*82*
String	**BARN** Unit of area.	*82*
String	**BECQUEREL** Unit of activity referred to a radionuclide.	*77*
String	**BEL** Unit of logarithmic ratio quantities.	*83*
String	**CANDELA** Unit of luminous intensity defined by the International System of Units (SI).	*72*
String	**CANDELA_PER_SQUARE_METRE** Unit of luminance.	*74*
String	**COULOMB** Unit of electronic charge, amount of electricity.	*75*
String	**COULOMB_PER_CUBIC_METRE** Unit of electric charge density.	*79*
String	**COULOMB_PER_KILOGRAM** Unit of exposure (x- and gamma-rays).	*80*
String	**COULOMB_PER_SQUARE_METRE** Unit of surface charge density, electric flux density, electric displacement.	*79*
String	**CUBIC_METRE** Unit of volume.	*73*
String	**CUBIC_METRE_PER_KILOGRAM** Unit of specific volume.	*73*
String	**DAY** Unit of time.	*81*
String	**DECIBEL** Unit of logarithmic ratio quantities.	*83*
String	**DEGREE** Unit of plane angle.	*81*

String	**DEGREE_CELSIUS** Unit of Celsius temperature.	*76*
String	**DYNE** Unit of force.	*83*
String	**ERG** Unit of energy.	*83*
String	**FARAD** Unit of capacitance.	*75*
String	**FARAD_PER_METRE** Unit of permittivity.	*79*
String	**GAL** Unit of acceleration.	*84*
String	**GAUSS** Unit of magnetic flux density.	*84*
String	**GRAY** Unit of absorbed dose, specific energy (imparted), kerma.	*77*
String	**GRAY_PER_SECOND** Unit of absorbed dose rate.	*80*
String	**HECTARE** Unit of area.	*81*
String	**HENRY** Unit of inductance.	*76*
String	**HENRY_PER_METRE** Unit of permeability.	*79*
String	**HERTZ** Unit of frequency.	*74*
String	**HOUR** Unit of time.	*81*
String	**JOULE** Unit of energy, work, amount of electricity.	*75*
String	**JOULE_PER_CUBIC_METRE** Unit of energy density.	*79*
String	**JOULE_PER_KELVIN** Unit of heat capacity, entropy.	*78*
String	**JOULE_PER_KILOGRAM** Unit of specific energy.	*78*
String	**JOULE_PER_KILOGRAM_KELVIN** Unit of specific heat capacity, specific entropy.	*78*
String	**JOULE_PER_MOLE** Unit of molar energy.	*80*
String	**JOULE_PER_MOLE_KELVIN** Unit of molar entropy, molar heat capacity.	*80*
String	**KATAL** Unit of catalytic activity.	*77*
String	**KATAL_PER_CUBIC_METRE** Unit of catalytic activity concentration.	*80*
String	**KELVIN** Unit of thermodynamic temperature defined by the International System of Units (SI).	*72*

`String`	**KILOGRAM** Unit of mass defined by the International System of Units (SI).	*72*
`String`	**KILOGRAM_PER_CUBIC_METRE** Unit of density, mass density, mass concentration.	*73*
`String`	**KILOGRAM_PER_SQUARE_METRE** Unit of surface density.	*73*
`String`	**KNOT** Unit of speed.	*83*
`String`	**LITRE** Unit of volume.	*82*
`String`	**LUMEN** Unit of luminous flux.	*76*
`String`	**LUX** Unit of illuminance.	*77*
`String`	**MAXWELL** Unit of magnetic flux.	*84*
`String`	**METRE** Unit of length defined by the International System of Units (SI).	*71*
`String`	**METRE_PER_SECOND** Unit of speed, velocity.	*73*
`String`	**METRE_PER_SECOND_SQUARED** Unit of acceleration.	*73*
`String`	**MILLIMETRE_OF_MERCURY** Unit of pressure.	*82*
`String`	**MOLE** Unit of amount of substance defined by the International System of Units (SI).	*72*
`String`	**MOLE_PER_CUBIC_METRE** Unit of amount concentration, concentration.	*74*
`String`	**NAUTICAL_MILE** Unit of distance.	*82*
`String`	**NEPER** Unit of logarithmic ratio quantities.	*83*
`String`	**NEWTON** Unit of force.	*75*
`String`	**NEWTON_METRE** Unit of moment of force.	*77*
`String`	**NEWTON_PER_METRE** Unit of surface tension.	*78*
`String`	**OERSTED** Unit of magnetic field.	*84*
`String`	**OHM** Unit of electric resistance.	*76*
`String`	**PASCAL** Unit of pressure, stress.	*75*
`String`	**PASCAL_SECOND** Unit of dynamic viscosity.	*77*
`String`	**PHOT** Unit of illuminance.	*84*

String	**PLANE_ANGLE_MINUTE** Unit of plane angle.	*81*
String	**PLANE_ANGLE_SECOND** Unit of plane angle.	*81*
String	**POISE** Unit of dynamic viscosity.	*83*
String	**PREFIX_ATTO** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*87*
String	**PREFIX_CENTI** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*86*
String	**PREFIX_DECA** Adopted prefix symbol to form the symbols of the decimal multiples of SI units.	*84*
String	**PREFIX_DECI** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*86*
String	**PREFIX_EXA** Adopted prefix symbol to form the symbols of the decimal multiples of SI units.	*85*
String	**PREFIX_FEMTO** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*86*
String	**PREFIX_GIGA** Adopted prefix symbol to form the symbols of the decimal multiples of SI units.	*85*
String	**PREFIX_HECTO** Adopted prefix symbol to form the symbols of the decimal multiples of SI units.	*85*
String	**PREFIX_KILO** Adopted prefix symbol to form the symbols of the decimal multiples of SI units.	*85*
String	**PREFIX_MEGA** Adopted prefix symbol to form the symbols of the decimal multiples of SI units.	*85*
String	**PREFIX_MICRO** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*86*
String	**PREFIX_MILLI** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*86*
String	**PREFIX_NANO** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*86*
String	**PREFIX_PICO** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*86*
String	**PREFIX_YOCTO** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*87*
String	**PREFIX_YOTTA** Adopted prefix symbol to form the symbols of the decimal multiples of SI units.	*85*
String	**PREFIX_ZEPTO** Adopted prefix symbol to form the symbols of the decimal submultiples of SI units.	*87*
String	**PREFIX_ZETTA** Adopted prefix symbol to form the symbols of the decimal multiples of SI units.	*85*
String	**RADIAN** Unit of plane angle.	*74*
String	**RADIAN_PER_SECOND** Unit of angular velocity.	*78*
String	**RADIAN_PER_SECOND_SQUARED** Unit of angular acceleration.	*78*

String	**RECIPROCAL_METRE** Unit of wavenumber.	*73*
String	**SECOND** Unit of time defined by the International System of Units (SI).	*72*
String	**SIEMENS** Unit of electric conductance.	*76*
String	**SIEVERT** Unit of dose equivalent, ambient dose equivalent, directional dose equivalent, personal dose equivalent.	*77*
String	**SQUARE_METRE** Unit of area.	*72*
String	**STERADIAN** Unit of solid angle.	*74*
String	**STILB** Unit of luminance.	*84*
String	**STOKES** Unit of kinematic viscosity.	*83*
String	**TESLA** Unit of magnetic flux density.	*76*
String	**TIME_MINUTE** Unit of time.	*81*
String	**TONNE** Unit of mass.	*82*
String	**VOLT** Unit of electric potential difference, electromotive force.	*75*
String	**VOLT_PER_METRE** Unit of electric field strength.	*79*
String	**WATT** Unit of power, radiant flux.	*75*
String	**WATT_PER_METRE_KELVIN** Unit of thermal conductivity.	*79*
String	**WATT_PER_SQUARE_METRE** Unit of heat flux density, irradiance.	*78*
String	**WATT_PER_SQUARE_METRE_STERADIAN** Unit of radiance.	*80*
String	**WATT_PER_STERADIAN** Unit of radiant intensity.	*80*
String	**WEBER** Unit of magnetic flux.	*76*

Field Detail

METRE

```
public static final String METRE = "m"
```

Unit of length defined by the International System of Units (SI). It's one of be base units called metre.

KILOGRAM

```
public static final String KILOGRAM = "kg"
```

> Unit of mass defined by the International System of Units (SI). It's one of be base units called kilogram.

SECOND

```
public static final String SECOND = "s"
```

> Unit of time defined by the International System of Units (SI). It's one of be base units called second.

AMPERE

```
public static final String AMPERE = "A"
```

> Unit of electric current defined by the International System of Units (SI). It's one of be base units called ampere.

KELVIN

```
public static final String KELVIN = "\u212a"
```

> Unit of thermodynamic temperature defined by the International System of Units (SI). It's one of be base units called kelvin.

MOLE

```
public static final String MOLE = "mol"
```

> Unit of amount of substance defined by the International System of Units (SI). It's one of be base units called mole.

CANDELA

```
public static final String CANDELA = "cd"
```

> Unit of luminous intensity defined by the International System of Units (SI). It's one of be base units called candela.

SQUARE_METRE

```
public static final String SQUARE_METRE = "m\u00b2"
```

> Unit of area. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called square metre.

CUBIC_METRE

```
public static final String CUBIC_METRE = "m\u00b3"
```

> Unit of volume. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called cubic metre.

METRE_PER_SECOND

```
public static final String METRE_PER_SECOND = "m/s"
```

> Unit of speed, velocity. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called metre per second.

METRE_PER_SECOND_SQUARED

```
public static final String METRE_PER_SECOND_SQUARED = "m/s\u00b2"
```

> Unit of acceleration. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called metre per second squared.

RECIPROCAL_METRE

```
public static final String RECIPROCAL_METRE = "m\u207b\u00b9"
```

> Unit of wavenumber. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called reciprocal metre.

KILOGRAM_PER_CUBIC_METRE

```
public static final String KILOGRAM_PER_CUBIC_METRE = "kg/m\u00b3"
```

> Unit of density, mass density, mass concentration. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called kilogram per cubic metre.

KILOGRAM_PER_SQUARE_METRE

```
public static final String KILOGRAM_PER_SQUARE_METRE = "kg/m\u00b2"
```

> Unit of surface density. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called kilogram per square metre.

CUBIC_METRE_PER_KILOGRAM

```
public static final String CUBIC_METRE_PER_KILOGRAM = "m\u00b3/kg"
```

> Unit of specific volume. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called cubic metre per kilogram.

AMPERE_PER_SQUARE_METRE

```
public static final String AMPERE_PER_SQUARE_METRE = "A/m\u00b2"
```

> Unit of current density. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called ampere per square metre.

AMPERE_PER_METRE

```
public static final String AMPERE_PER_METRE = "A/m"
```

> Unit of magnetic field strength. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called ampere per metre.

MOLE_PER_CUBIC_METRE

```
public static final String MOLE_PER_CUBIC_METRE = "mol/m\u00b3"
```

> Unit of amount concentration, concentration. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called mole per cubic metre.

CANDELA_PER_SQUARE_METRE

```
public static final String CANDELA_PER_SQUARE_METRE = "cd/m\u00b2"
```

> Unit of luminance. It's one of coherent derived units in the SI expressed in terms of base units. The unit is called candela per square metre.

RADIAN

```
public static final String RADIAN = "rad"
```

> Unit of plane angle. It's one of the coherent derived units in the SI with special names and symbols. The unit is called radian.

STERADIAN

```
public static final String STERADIAN = "sr"
```

> Unit of solid angle. It's one of the coherent derived units in the SI with special names and symbols. The unit is called steradian.

HERTZ

```
public static final String HERTZ = "Hz"
```

> Unit of frequency. It's one of the coherent derived units in the SI with special names and symbols. The unit is called hertz.

NEWTON

```
public static final String NEWTON = "N"
```

Unit of force. It's one of the coherent derived units in the SI with special names and symbols. The unit is called newton.

PASCAL

```
public static final String PASCAL = "Pa"
```

Unit of pressure, stress. It's one of the coherent derived units in the SI with special names and symbols. The unit is called pascal.

JOULE

```
public static final String JOULE = "J"
```

Unit of energy, work, amount of electricity. It's one of the coherent derived units in the SI with special names and symbols. The unit is called joule.

WATT

```
public static final String WATT = "W"
```

Unit of power, radiant flux. It's one of the coherent derived units in the SI with special names and symbols. The unit is called watt.

COULOMB

```
public static final String COULOMB = "C"
```

Unit of electronic charge, amount of electricity. It's one of the coherent derived units in the SI with special names and symbols. The unit is called coulomb.

VOLT

```
public static final String VOLT = "V"
```

Unit of electric potential difference, electromotive force. It's one of the coherent derived units in the SI with special names and symbols. The unit is called volt.

FARAD

```
public static final String FARAD = "F"
```

Unit of capacitance. It's one of the coherent derived units in the SI with special names and symbols. The unit is called farad.

OHM

```
public static final String OHM = "\u2126"
```

Unit of electric resistance. It's one of the coherent derived units in the SI with special names and symbols. The unit is called ohm.

SIEMENS

```
public static final String SIEMENS = "S"
```

Unit of electric conductance. It's one of the coherent derived units in the SI with special names and symbols. The unit is called siemens.

WEBER

```
public static final String WEBER = "Wb"
```

Unit of magnetic flux. It's one of the coherent derived units in the SI with special names and symbols. The unit is called weber.

TESLA

```
public static final String TESLA = "T"
```

Unit of magnetic flux density. It's one of the coherent derived units in the SI with special names and symbols. The unit is called tesla.

HENRY

```
public static final String HENRY = "H"
```

Unit of inductance. It's one of the coherent derived units in the SI with special names and symbols. The unit is called henry.

DEGREE_CELSIUS

```
public static final String DEGREE_CELSIUS = "\u2103"
```

Unit of Celsius temperature. It's one of the coherent derived units in the SI with special names and symbols. The unit is called degree Celsius.

LUMEN

```
public static final String LUMEN = "lm"
```

Unit of luminous flux. It's one of the coherent derived units in the SI with special names and symbols. The unit is called lumen.

LUX

```
public static final String LUX = "lx"
```

Unit of illuminance. It's one of the coherent derived units in the SI with special names and symbols. The unit is called lux.

BECQUEREL

```
public static final String BECQUEREL = "Bq"
```

Unit of activity referred to a radionuclide. It's one of the coherent derived units in the SI with special names and symbols. The unit is called becquerel.

GRAY

```
public static final String GRAY = "Gy"
```

Unit of absorbed dose, specific energy (imparted), kerma. It's one of the coherent derived units in the SI with special names and symbols. The unit is called gray.

SIEVERT

```
public static final String SIEVERT = "Sv"
```

Unit of dose equivalent, ambient dose equivalent, directional dose equivalent, personal dose equivalent. It's one of the coherent derived units in the SI with special names and symbols. The unit is called sievert.

KATAL

```
public static final String KATAL = "kat"
```

Unit of catalytic activity. It's one of the coherent derived units in the SI with special names and symbols. The unit is called katal.

PASCAL_SECOND

```
public static final String PASCAL_SECOND = "Pa s"
```

Unit of dynamic viscosity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called pascal second.

NEWTON_METRE

```
public static final String NEWTON_METRE = "N m"
```

Unit of moment of force. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called newton metre.

NEWTON_PER_METRE

`public static final String` **`NEWTON_PER_METRE`** `= "N/m"`

> Unit of surface tension. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called newton per metre.

RADIAN_PER_SECOND

`public static final String` **`RADIAN_PER_SECOND`** `= "rad/s"`

> Unit of angular velocity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called radian per second.

RADIAN_PER_SECOND_SQUARED

`public static final String` **`RADIAN_PER_SECOND_SQUARED`** `= "rad/s\u00b2"`

> Unit of angular acceleration. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called radian per second squared.

WATT_PER_SQUARE_METRE

`public static final String` **`WATT_PER_SQUARE_METRE`** `= "W/m\u00b2"`

> Unit of heat flux density, irradiance. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called watt per square metre.

JOULE_PER_KELVIN

`public static final String` **`JOULE_PER_KELVIN`** `= "J/K"`

> Unit of heat capacity, entropy. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per kelvin.

JOULE_PER_KILOGRAM_KELVIN

`public static final String` **`JOULE_PER_KILOGRAM_KELVIN`** `= "J/(kg K)"`

> Unit of specific heat capacity, specific entropy. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per kilogram kelvin.

JOULE_PER_KILOGRAM

`public static final String` **`JOULE_PER_KILOGRAM`** `= "J/kg"`

> Unit of specific energy. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per kilogram.

WATT_PER_METRE_KELVIN

public static final String **WATT_PER_METRE_KELVIN** = "W/(m K)"

Unit of thermal conductivity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called watt per metre kelvin.

JOULE_PER_CUBIC_METRE

public static final String **JOULE_PER_CUBIC_METRE** = "J/m\u00b3"

Unit of energy density. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per cubic metre.

VOLT_PER_METRE

public static final String **VOLT_PER_METRE** = "V/m"

Unit of electric field strength. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called volt per metre.

COULOMB_PER_CUBIC_METRE

public static final String **COULOMB_PER_CUBIC_METRE** = "C/m\u00b3"

Unit of electric charge density. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called coulomb per cubic metre.

COULOMB_PER_SQUARE_METRE

public static final String **COULOMB_PER_SQUARE_METRE** = "C/m\u00b2"

Unit of surface charge density, electric flux density, electric displacement. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called coulomb per square metre.

FARAD_PER_METRE

public static final String **FARAD_PER_METRE** = "F/m"

Unit of permittivity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called farad per metre.

HENRY_PER_METRE

public static final String **HENRY_PER_METRE** = "H/m"

Unit of permeability. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called henry per metre.

JOULE_PER_MOLE

`public static final String` **`JOULE_PER_MOLE`** `= "J/mol"`

Unit of molar energy. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per mole.

JOULE_PER_MOLE_KELVIN

`public static final String` **`JOULE_PER_MOLE_KELVIN`** `= "J/(mol K)"`

Unit of molar entropy, molar heat capacity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called joule per mole kelvin.

COULOMB_PER_KILOGRAM

`public static final String` **`COULOMB_PER_KILOGRAM`** `= "C/kg"`

Unit of exposure (x- and gamma-rays). It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called coulomb per kilogram.

GRAY_PER_SECOND

`public static final String` **`GRAY_PER_SECOND`** `= "Gy/s"`

Unit of absorbed dose rate. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called gray per second.

WATT_PER_STERADIAN

`public static final String` **`WATT_PER_STERADIAN`** `= "W/sr"`

Unit of radiant intensity. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called watt per steradian.

WATT_PER_SQUARE_METRE_STERADIAN

`public static final String` **`WATT_PER_SQUARE_METRE_STERADIAN`** `= "W/(m\u00b2 sr)"`

Unit of radiance. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called watt per square metre steradian.

KATAL_PER_CUBIC_METRE

`public static final String` **`KATAL_PER_CUBIC_METRE`** `= "kat/m\u00b3"`

Unit of catalytic activity concentration. It's one of coherent derived units whose names and symbols include SI coherent derived units with special names and symbols. The unit is called katal per cubic metre.

TIME_MINUTE

```
public static final String TIME_MINUTE = "min"
```

Unit of time. It's one of non-SI units accepted for use with the International System of Units. The unit is called minute.

HOUR

```
public static final String HOUR = "h"
```

Unit of time. It's one of non-SI units accepted for use with the International System of Units. The unit is called hour.

DAY

```
public static final String DAY = "d"
```

Unit of time. It's one of non-SI units accepted for use with the International System of Units. The unit is called day.

DEGREE

```
public static final String DEGREE = "\u00b0"
```

Unit of plane angle. It's one of non-SI units accepted for use with the International System of Units. The unit is called degree.

PLANE_ANGLE_MINUTE

```
public static final String PLANE_ANGLE_MINUTE = "\u2032"
```

Unit of plane angle. It's one of non-SI units accepted for use with the International System of Units. The unit is called minute.

PLANE_ANGLE_SECOND

```
public static final String PLANE_ANGLE_SECOND = "\u2033"
```

Unit of plane angle. It's one of non-SI units accepted for use with the International System of Units. The unit is called second.

HECTARE

```
public static final String HECTARE = "ha"
```

Unit of area. It's one of non-SI units accepted for use with the International System of Units. The unit is called hectare.

LITRE

```
public static final String LITRE = "l"
```

Unit of volume. It's one of non-SI units accepted for use with the International System of Units. The unit is called litre. International System of Units accepts two symbols: lower-case l and capital L. That constant value is using the lower-case l.

TONNE

```
public static final String TONNE = "t"
```

Unit of mass. It's one of non-SI units accepted for use with the International System of Units. The unit is called tonne.

BAR

```
public static final String BAR = "bar"
```

Unit of pressure. It's one of other non-SI units. The unit is called bar.

MILLIMETRE_OF_MERCURY

```
public static final String MILLIMETRE_OF_MERCURY = "mmHg"
```

Unit of pressure. It's one of other non-SI units. The unit is called millimetre of mercury.

ANGSTROM

```
public static final String ANGSTROM = "\u212b"
```

Unit of length. It's one of other non-SI units. The unit is called angstrom.

NAUTICAL_MILE

```
public static final String NAUTICAL_MILE = "M"
```

Unit of distance. It's one of other non-SI units. The unit is called nautical mile.

BARN

```
public static final String BARN = "b"
```

Unit of area. It's one of other non-SI units. The unit is called barn.

KNOT

```
public static final String KNOT = "kn"
```

Unit of speed. It's one of other non-SI units. The unit is called knot.

NEPER

```
public static final String NEPER = "Np"
```

Unit of logarithmic ratio quantities. It's one of other non-SI units. The unit is called neper.

BEL

```
public static final String BEL = "B"
```

Unit of logarithmic ratio quantities. It's one of other non-SI units. The unit is called bel.

DECIBEL

```
public static final String DECIBEL = "dB"
```

Unit of logarithmic ratio quantities. It's one of other non-SI units. The unit is called decibel.

ERG

```
public static final String ERG = "erg"
```

Unit of energy. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called erg.

DYNE

```
public static final String DYNE = "dyn"
```

Unit of force. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called dyne.

POISE

```
public static final String POISE = "P"
```

Unit of dynamic viscosity. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called poise.

STOKES

```
public static final String STOKES = "St"
```

Unit of kinematic viscosity. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called stokes.

STILB

```
public static final String STILB = "sb"
```

Unit of luminance. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called stilb.

PHOT

```
public static final String PHOT = "ph"
```

Unit of illuminance. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called phot.

GAL

```
public static final String GAL = "Gal"
```

Unit of acceleration. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called gal.

MAXWELL

```
public static final String MAXWELL = "Mx"
```

Unit of magnetic flux. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called maxwell.

GAUSS

```
public static final String GAUSS = "G"
```

Unit of magnetic flux density. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called gauss.

OERSTED

```
public static final String OERSTED = "Oe"
```

Unit of magnetic field. It's one of non-SI units associated with the CGS and the CGS-Gaussian system of units. The unit is called oersted.

PREFIX_DECA

```
public static final String PREFIX_DECA = "da"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called deca and represents the 1st power of ten.

PREFIX_HECTO

```
public static final String PREFIX_HECTO = "h"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called hecto and represents the 2nd power of ten.

PREFIX_KILO

```
public static final String PREFIX_KILO = "k"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called kilo and represents the 3rd power of ten.

PREFIX_MEGA

```
public static final String PREFIX_MEGA = "M"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called mega and represents the 6th power of ten.

PREFIX_GIGA

```
public static final String PREFIX_GIGA = "G"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called giga and represents the 9th power of ten.

PREFIX_EXA

```
public static final String PREFIX_EXA = "E"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called exa and represents the 18th power of ten.

PREFIX_ZETTA

```
public static final String PREFIX_ZETTA = "Z"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called zetta and represents the 21th power of ten.

PREFIX_YOTTA

```
public static final String PREFIX_YOTTA = "Y"
```

Adopted prefix symbol to form the symbols of the decimal multiples of SI units. It's called yotta and represents the 24th power of ten.

PREFIX_DECI

```
public static final String PREFIX_DECI = "d"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called deci and represents the 1st negative power of ten.

PREFIX_CENTI

```
public static final String PREFIX_CENTI = "c"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called centi and represents the 2nd negative power of ten.

PREFIX_MILLI

```
public static final String PREFIX_MILLI = "m"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called milli and represents the 3rd negative power of ten.

PREFIX_MICRO

```
public static final String PREFIX_MICRO = "\u00b5"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called micro and represents the 6th negative power of ten.

PREFIX_NANO

```
public static final String PREFIX_NANO = "n"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called nano and represents the 9th negative power of ten.

PREFIX_PICO

```
public static final String PREFIX_PICO = "p"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called pico and represents the 12th negative power of ten.

PREFIX_FEMTO

```
public static final String PREFIX_FEMTO = "f"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called femto and represents the 15th negative power of ten.

PREFIX_ATTO

```
public static final String PREFIX_ATTO = "a"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called atto and represents the 18th negative power of ten.

PREFIX_ZEPTO

```
public static final String PREFIX_ZEPTO = "z"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called zepto and represents the 21th negative power of ten.

PREFIX_YOCTO

```
public static final String PREFIX_YOCTO = "y"
```

Adopted prefix symbol to form the symbols of the decimal submultiples of SI units. It's called yocto and represents the 24th negative power of ten.

Java API documentation generated with **DocFlex/Doclet** v1.5.6

DocFlex/Doclet is both a multi-format Javadoc doclet and a free edition of **DocFlex/Javadoc**. If you need to customize your Javadoc without writing a full-blown doclet from scratch, DocFlex/Javadoc may be the only tool able to help you! Find out more at **www.docflex.com**

8 Considered Alternatives

8.1 Use Configuration Admin to update the Device service properties

OSGi service properties are used to represent the Device service properties. The properties can be updated with the help of `org.osgi.framework.ServiceRegistration.setProperties(Dictionary)` method. The service registration is intended for a private usage and should not be shared between the bundles.

The current design provides set methods, which can be used when an external application wants to modify the Device service properties. It's simple and a part of Device interface. We have to define a new permission check, because there is no such protection to `org.osgi.framework.ServiceRegistration.setProperties` method.

Considered alternative was about property update based on configuration update in the Configuration Admin service. The Device service properties can be updated when the corresponding configuration properties are updated. The disadvantages here are:

- Device properties duplication – they are stored in the device configuration and in the Device service properties.

- Possible performance issue when a lot of devices are used.

8.2 DeviceAdmin interface availability

DeviceAdmin service was removed from the current RFC document. That management functionality can be provided by a different specification document. That considered alternative is kept for completeness.

DeviceAdmin service can simplify the device service registration. It hides the implementation details i.e. realize program to an interface rather than to an implementation.

The considered alternative is not to use that interface and to register the Device service implementation to the OSGi service registry. Here are two code snippets, which demonstrates positives and negatives:

1. Without DeviceAdmin

```
Map ipCameraProps = new HashMap(3, 1F);

ipCameraProps.put("IP.Camera.Address", "192.168.0.21");

ipCameraProps.put("IP.Camera.Username", "test");

ipCameraProps.put("IP.Camera.Password", "test");


//WARNING - an access to implementation class, which should be bundle private

IPCameraDeviceImpl ipCameraImpl = new IPCameraDeviceImpl(ipCameraProps);

ipCameraImpl.register(bundleContext);

// play the video stream...
```

```
// remove the device

ipCameraImpl.unregister();
```

That snippet demonstrate program to implementation rather than an interface, which break basic OOP rule.

2. With DeviceAdmin

```
Map ipCameraProps = new HashMap(3, 1F);

ipCameraProps.put("IP.Camera.Address", "192.168.0.21");

ipCameraProps.put("IP.Camera.Username", "test");

ipCameraProps.put("IP.Camera.Password", "test");


DeviceAdmin ipCameraDeviceAdmin = getIPCameraDeviceAdmin();

Device ipCamera = ipCameraDeviceAdmin.add(ipCameraProps);

// play the device video stream

// remove the device

ipCamera.remove();
```

It demonstrate program to interface rather than an implementation, which is the correct approach.

8.3   Access helper methods removal of FunctionalDevice

org.osgi.service.functionaldevice.FunctionalDevice.getChildren(), org.osgi.service.functionaldevice.FunctionalDevice.getParent() and org.osgi.service.functionaldevice.FunctionalDevice.getReferences() were removed, because they provided access to the FunctionalDevice services outside the OSGi service registry. It can be problematic in various scenarios like:

• The service Find Hook can be ignored.

• No service unget is possible for such shared service instances.

• The dependency tools based on the service registry cannot track such sharings.

9   Security Considerations

9.1   Device Permission

The device permission controls the bundle's authority to perform specific privileged administrative operations on the devices. The action for this permission is:

Action	Method
ACTION_REMOVE	Device.remove()

The name of the permission is a filter based. For more details about filter based permissions, see OSGi Core Specification, Filter Based Permissions. The filter provides an access to all device service properties. The service property names are case insensitive. The filter attribute names are processed in a case insensitive manner. For example, the operator can give a bundle the permission to only manage devices of vendor "acme":

```
org.osgi.service.dal.DevicePermission("dal.device.hardware.vendor=acme", ...)
```

The permission action allows the operator to assign only the necessary permissions to the bundle. For example, the management bundle can have permission to remove all registered devices:

```
org.osgi.service.dal.DevicePermission("*", "remove")
```

The code that needs to check the device permission must always use the constructor that takes the device as a parameter `DevicePermission(Device, String)` with a single action. For example, the implementation of `org.osgi.service.dal.Device.remove()` method must check that the caller has an access to the operation:

```
public class DeviceImpl implements Device {

    public void start() {

        securityManager.checkPermission(new DevicePermission(this, "remove"));

    }

}
```

9.2   Required Permissions

The Functional Device implementation must check the caller for the appropriate Functional Device Permission before execution of the real operation actions like remove. Once the Functional Device Permission is checked against the caller the implementation will proceed with the actual operation. The operation can require a number of other permissions to complete. The implementation must isolate the caller from such permission checks by use of proper privileged blocks.

10 Document Support

10.1 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].   JavaBeans Spec, http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html

[4].   Unicode Standard Annex #15, Unicode Normalization Forms

10.2 Author's Address

Name	Evgeni Grigorov
Company	ProSyst Software
Address	Aachenerstr. 222, 50935 Cologne, Germany
Voice	+49 221 6604 501
e-mail	e.grigorov@prosyst.com

10.3 Acronyms and Abbreviations

Item	Description
Device Abstraction Layer	Unifies the work with devices provided by different protocols.
Device Abstraction API	Unified API for management of devices provided by different protocols.
Device Abstraction Adapter	Examples for such adapters are ZigBee Adapter, Z-Wave Adapter etc. Provides support for a particular device protocol to Device Abstraction Layer. The adapter integrates the protocol specific driver devices.

10.4 End of Document