Device Abstraction Layer Functions

Draft

68 Pages

Abstract

Defines a core set of functions to RFC-0196 Device Abstraction Layer. They provide an interoperability between the different specific domains.

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	Feb 21 2013	Detach the core functions from RFC-0196 Device Abstraction Layer. Evgeni Grigorov, ProSyst Software, *e.grigorov@prosyst.com*

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 core set of functions to RFC-0196 Device Abstraction Layer. They provide an interoperability between the different specific domains. The set can be reused, extended or fully replaced into specific domain.

2 Application Domain

OSGi Device Abstraction Layer RFC-0196 don't define functions, but their common representation. It doesn't guarantee interoperability between the different domains. In this way, the same functionality can be modeled in different ways and the applications will be bind to the specific interfaces.

For example, a meter function can be defined in the smart home domain as an energy meter, but in the security domain as a time meter (timer).

In order to unify the application access to the basic functionalities, a core of functions is required.

3 Problem Description

The applications need to know about a set of OSGi Device Abstraction Layer functions to operate with them. They can execute operations, set and receive property values.

Illustration 1 shows one possible approach for working with heterogeneous functions, which are related to the same functionality. The smart home meter and vendor specific meter are all about the same kind of information. They collect metering information.

In this case each application must use specific API for this function. One obvious disadvantage of this model is that when a new function type is added the applications must be modified in order to support it.

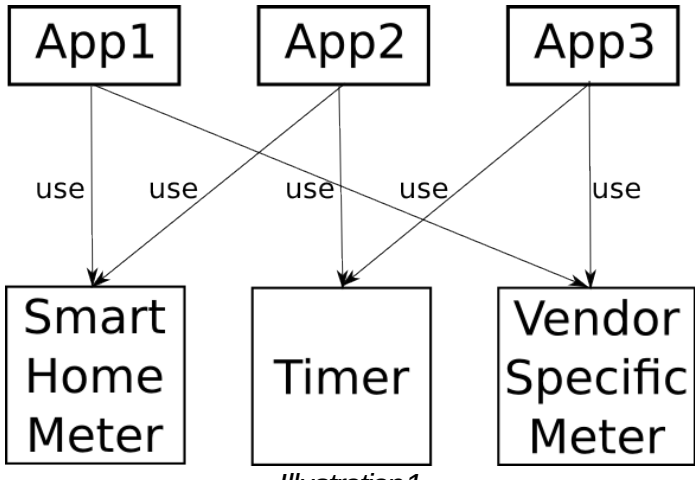

Illustration 1

Much better is the approach from Illustration 2 which is defined by this RFC.

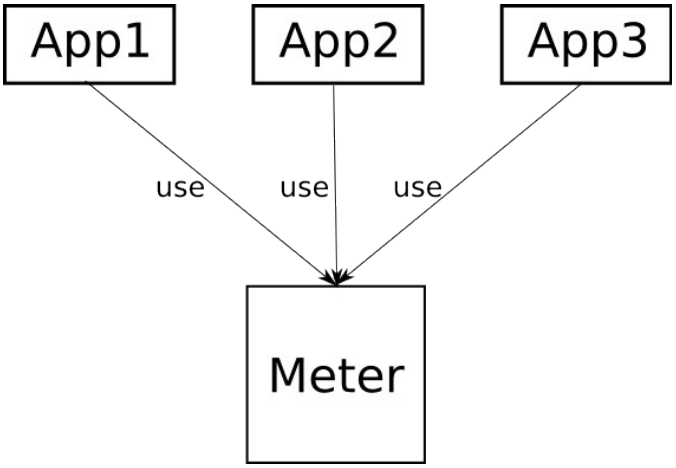

Illustration2

In this case a core functions are introduced to ensure the interoperability between the different domains. Thus the following advantages are achieved:

- The application programmers can work with the same set of functions.
- The functions can be reused or extended.
- The application can work without modifications when a new vendor function types are registered.

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

Requirement 5. 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.

5 Technical Solution

5.1 Core Functions

Concrete function interfaces have to be defined to unify the access and control of the basic operations and related properties. The current section specifies the minimal basic set of such functionality. It can be reused and extended to cover more specific scenarios. They are about the control, monitoring and metering information.

5.1.1 BooleanControl Function

`BooleanControl` function provides a binary control support. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The full function definition is available in the next table.

BooleanControl	
Name	**Description**
Operations	
reverse	Reverses the `BooleanControl` state. If the current state represents `true` value, it'll be reversed to `false`. If the current state represents `false` value, it'll be reversed to `true`.
setTrue	Sets the `BooleanControl` state to `true` value.
setFalse	Sets the `BooleanControl` state to `false` value.
Properties	
data	Contains the current state of `BooleanControl`. The property access can be: readable, writable and eventable.
Types	
light, door, window, power, other type defined in `org.osgi.service.dal.functions.Types` or vendor specific type.	

`BooleanData` data structure is used to provide information about the function state. That data object contains the boolean value, the value collecting time and additional metadata. The immutable `BooleanData.value` field is accessible with `BooleanData.getValue()` getter.

The function class diagram is depicted on Illustration 3. The next code snippet sets to `true` all `BooleanControl` functions.

```
final ServiceReference[] booleanControlSRefs = context.getServiceReferences(
      BooleanControl.class.getName(), null);
if (null == booleanControlSRefs) {
  return; // no such services
}
```

```
for (int i = 0; i < booleanControlSRefs.length; i++) {
  final BooleanControl booleanControl = (BooleanControl) context.getService(
      binaryControlSRefs[i]);
  if (null != booleanControl) {
    booleanControl.setTrue();
  }
}
```

5.1.2 BooleanSensor Function

`BooleanSensor` function provides binary sensor monitoring. It reports its state when an important event is available. There are no operations. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The full function definition is available in the next table.

BooleanSensor	
Name	**Description**
Properties	
data	Contains the current state of `BooleanSensor`. The property access can be: readable and eventable.
Types	
light, gas, smoke, door, window, power, rain, contact, fire, occupancy, water, motion, other type defined in `org.osgi.service.dal.functions.Types` or vendor specific type.	

`BooleanSensor` and Boolean`Control` are using the same Boolean`Data` data structure to provide information about the state. For more details see the definition in BooleanControl Function. The function class diagram is depicted on Illustration 3.

5.1.3 MultiLevelControl Function

`MultiLevelControl` function provides multi-level control support. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The full function definition is available in the next table.

MultiLevelControl	
Name	**Description**
Properties	
data	Contains the current state of `MultiLevelControl`. The property access can be: readable, writable and eventable.
Types	
light, temperature, flow, pressure, humidity, gas, smoke, door, window, liquid, power, noisiness, other type defined in `org.osgi.service.dal.functions.Types` or vendor specific type.	

`LevelData` data structure is used to provide information about the function level. That data object contains the `BigDecimal` value and the value unit. The measurement unit is used as it's defined in RFC-0196 OSGi Device

Abstraction Later. The immutable `LevelData.unit` field is accessible with `LevelData.getUnit()` getter. The immutable `LevelData.level` field is accessible with `LevelData.getLevel()` getter.

The function class diagram is depicted on Illustration 3.

5.1.4 MultiLevelSensor Function

`MultiLevelSensor` function provides multi-level sensor monitoring. It reports its state when an important event is available. There are no operations. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The full function definition is available in the next table.

MultiLevelSensor	
Name	**Description**
Properties	
data	Contains the current state of `MultiLevelSensor`. The property access can be: readable and eventable.
Types	
light, temperature, flow, pressure, humidity, gas, smoke, door, window, liquid, power, noisiness, rain, other type defined in `org.osgi.service.dal.functions.Types` or vendor specific type.	

`MultiLevelSensor` and `MultiLevelControl` are using the same `LevelData` data structure to provide information about the level. For more details see the definition in MultiLevelControl Function. The function class diagram is depicted on Illustration 3.

5.1.5 Meter Function

`Meter` function can measure metering information.

Meter	
Name	**Description**
Operations	
resetTotal	Resets the total metering info.
Properties	
total	Contains the total consumption. It has been measured since the last call of resetTotal or device initial run. The property access is readable.
current	Contains the current consumption. The property is readable.
Service Properties	
dal.meter.flow	Contains the metering flow. Currently, it can be “in” and “out”.
Types	
pressure, gas, power, water, heat, cold, other type defined in `org.osgi.service.dal.functions.Types` or vendor specific type.	

`Meter` function is using the same `LevelData` data structure as `MultiLevelSensor` and `MultiLevelControl` to provide metering information. For more details see the definition in MultiLevelControl Function. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later. The function class diagram is depicted on Illustration 3.

5.1.6 Alarm Function

`Alarm` function provides alarm sensor support. There is only one eventable property and no operations. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later.

Alarm	
Name	**Description**
Properties	
alarm	Specifies the alarm property name. The property is eventable.

`AlarmData` data structure is used to provide information about the available alarm. That data object contains the alarm type and severity.

The function class diagram is depicted on Illustration 3.

5.1.7 Keypad Function

`Keypad` function provides support for keypad control. A keypad typically consists of one or more keys/buttons, which can be discerned. Different types of key presses like short and long press can typically also be detected. There is only one eventable property and no operations. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later.

Keypad	
Name	**Description**
Properties	
key	Specifies a property name for a key from the keypad. The property is eventable.

`KeypadData` data structure is used to provide information when a change with some key from device keypad has occurred. That data object contains the event type, key code and key name. Currently, there are a few predefined event types:

- `EVENT_TYPE_PRESSED` – used for a key pressed;
- `EVENT_TYPE_PRESSED_LONG` – used for a long key pressed;
- `EVENT_TYPE_PRESSED_DOUBLE` – used for a double key pressed;
- `EVENT_TYPE_PRESSED_DOUBLE_LONG` – used for a double and long key pressed;
- `EVENT_TYPE_RELEASED` – used for a key released.
- `EVENT_TYPE_UNKNOWN` – represents an unknown keypad event type.

The function class diagram is depicted on Illustration 3.

5.1.8 WakeUp Function

`WakeUp` function provides device awake monitoring and management. It's especially applicable to battery-operated devices. Such device can notify the system that it's awake and can receive commands with an event to property `PROPERTY_AWAKE`. The property eventing must follow the definition in RFC-0196 OSGi Device Abstraction Later.

The device can periodically wake up for commands. The interval can be managed with `PROPERTY_WAKE_UP_INTERVAL` property.

The application can minimize the power consumption with `sleep()` operation. As a result, the device will sleep and will not receive commands to the next awake.

WakeUp	
Name	**Description**
Properties	
awake	Specifies the awake eventable property name. If the device is awake, it will trigger a property event. The property value type is `BooleanData`.
wakeUpInterval	Specifies the wake up interval. The device can periodically wake up and receive commands. That interval is managed by this property. The property can be readable and writable. The property value type is `LevelData`.
Operations	
sleep	The device is forced to sleep to minimize the power consumption.

The function class diagram is depicted on Illustration 3.

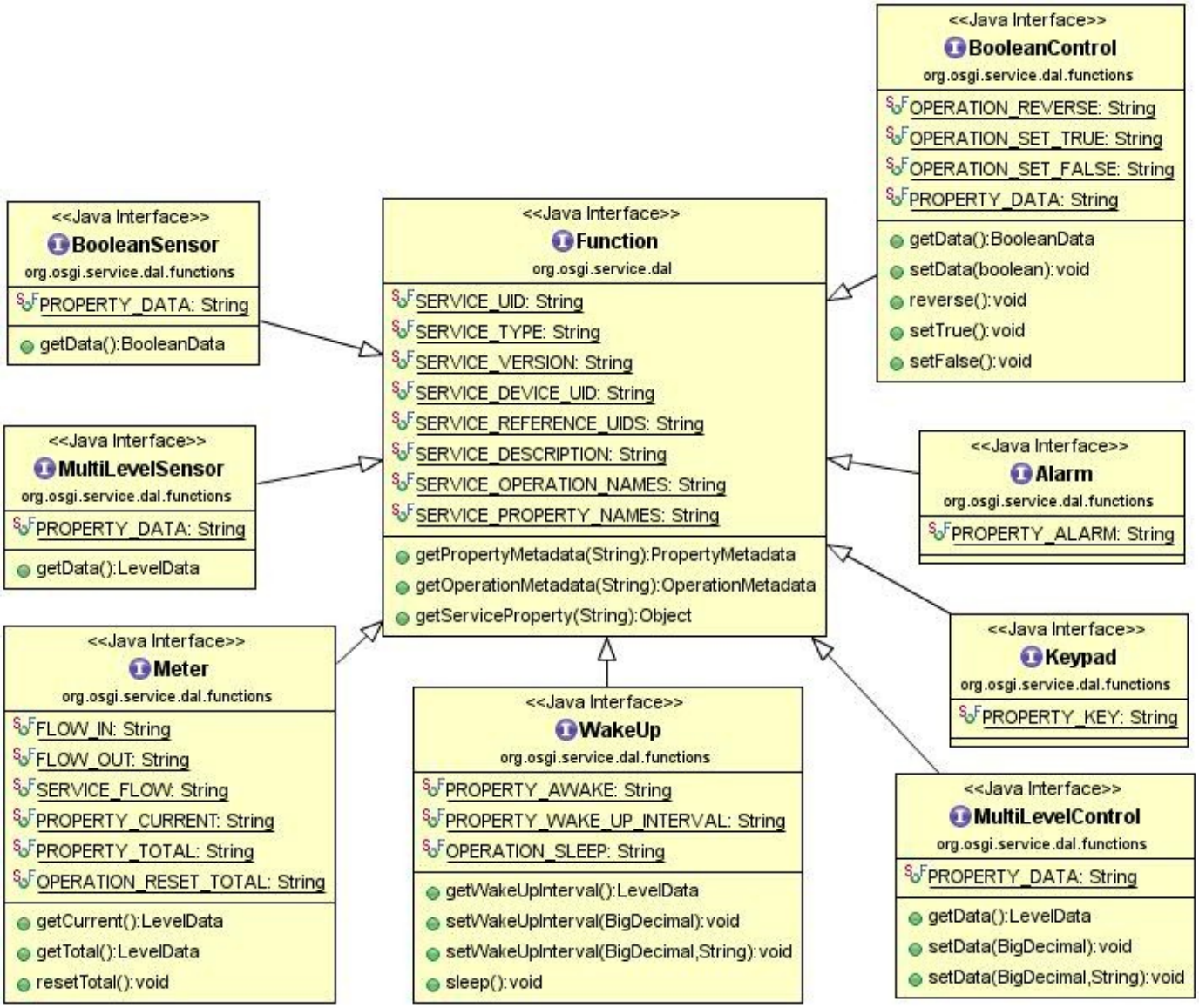

Illustration3

6 Data Transfer Objects

TODO: Do we need those objects?

7 Javadoc

OSGi Javadoc

2/24/14 10:32 AM

Package Summary		Page
org.osgi.service.dal.functions	Functions 1.0.	15
org.osgi.service.dal.functions.data	Function Data 1.0.	45

Package org.osgi.service.dal.functions

Functions 1.0.

See:

Description

Interface Summary		Page
Alarm	`Alarm` function provides alarm sensor support.	16
BooleanControl	`BooleanControl` function provides a boolean control support.	17
BooleanSensor	`BooleanSensor` function provides boolean sensor monitoring.	21
Keypad	`Keypad` function provides support for keypad control.	23
Meter	`Meter` function can measure metering information.	24
MultiLevelControl	`MultiLevelControl` function provides multi-level control support.	28
MultiLevelSensor	`MultiLevelSensor` function provides multi-level sensor monitoring.	31
Types	Shares common constants for all functions defined in this package.	33
WakeUp	`WakeUp` function provides device awake monitoring and management.	42

Package org.osgi.service.dal.functions Description

Functions 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.functions; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

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

Interface Alarm

org.osgi.service.dal.functions

All Superinterfaces:

org.osgi.service.dal.Function

```
public interface Alarm
extends org.osgi.service.dal.Function
```

`Alarm` function provides alarm sensor support. There is only one eventable property and no operations.

See Also:

AlarmData

Field Summary		Page
String	**PROPERTY_ALARM** Specifies the alarm property name.	16

Fields inherited from interface org.osgi.service.dal.Function
SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Methods inherited from interface org.osgi.service.dal.Function
getOperationMetadata, getPropertyMetadata, getServiceProperty

Field Detail

PROPERTY_ALARM

```
public static final String PROPERTY_ALARM = "alarm"
```

Specifies the alarm property name. The property is eventable.

See Also:

AlarmData

Interface BooleanControl

org.osgi.service.dal.functions

All Superinterfaces:

org.osgi.service.dal.Function

```
public interface BooleanControl
extends org.osgi.service.dal.Function
```

`BooleanControl` function provides a boolean control support. The function state is accessible with `getData()` getter and `setData(boolean)` setter. The state can be reversed with `reverse()` method, can be set to `true` value with `setTrue()` method and can be set to `false` value with `setFalse()` method.

As an example, the function is easily mappable to ZigBee OnOff cluster and Z-Wave Binary Switch command class. The control type can be:

- `Types.TYPE_LIGHT`
- `Types.TYPE_DOOR`
- `Types.TYPE_WINDOW`
- `Types.TYPE_POWER`
- other type defined in `Types`
- custom - vendor specific type

See Also:

`BooleanData`

Field Summary		Page
String	**OPERATION_REVERSE** Specifies the reverse operation name.	18
String	**OPERATION_SET_FALSE** Specifies the operation name, which sets the control state to `false` value.	18
String	**OPERATION_SET_TRUE** Specifies the operation name, which sets the control state to `true` value.	18
String	**PROPERTY_DATA** Specifies the state property name.	18

Fields inherited from interface org.osgi.service.dal.Function
SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Method Summary		Page
BooleanData	**getData**() Returns the current state of `BooleanControl`.	18
void	**reverse**() Reverses the `BooleanControl` state.	19
void	**setData**(boolean data) Sets the `BooleanControl` state to the specified value.	19
void	**setFalse**() Sets the `BooleanControl` state to `false` value.	20

void	**setTrue**() Sets the `BooleanControl` state to `true` value.	19

Methods inherited from interface org.osgi.service.dal.Function
`getOperationMetadata, getPropertyMetadata, getServiceProperty`

Field Detail

OPERATION_REVERSE

```
public static final String OPERATION_REVERSE = "reverse"
```

Specifies the reverse operation name. The operation can be executed with `reverse()` method.

OPERATION_SET_TRUE

```
public static final String OPERATION_SET_TRUE = "setTrue"
```

Specifies the operation name, which sets the control state to `true` value. The operation can be executed with `setTrue()` method.

OPERATION_SET_FALSE

```
public static final String OPERATION_SET_FALSE = "setFalse"
```

Specifies the operation name, which sets the control state to `false` value. The operation can be executed with `setFalse()` method.

PROPERTY_DATA

```
public static final String PROPERTY_DATA = "data"
```

Specifies the state property name. The property value is accessible with `getData()` method.

See Also:
`BooleanData`

Method Detail

getData

```
BooleanData getData()
            throws UnsupportedOperationException,
                   IllegalStateException,
                   org.osgi.service.dal.DeviceException
```

Returns the current state of `BooleanControl`. It's a getter method for `PROPERTY_DATA` property.

Returns:
The current state of `BooleanControl`.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

See Also:

`BooleanData`, `PROPERTY_DATA`

---

setData

```
void setData(boolean data)
      throws UnsupportedOperationException,
             IllegalStateException,
             org.osgi.service.dal.DeviceException,
             IllegalArgumentException
```

Sets the `BooleanControl` state to the specified value. It's setter method for `PROPERTY_DATA` property.

Parameters:

`data` - The new function value.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.
`IllegalArgumentException` - If there is an invalid argument.

See Also:

`PROPERTY_DATA`

---

reverse

```
void reverse()
      throws UnsupportedOperationException,
             IllegalStateException,
             org.osgi.service.dal.DeviceException
```

Reverses the `BooleanControl` state. If the current state represents `true` value, it'll be reversed to `false`. If the current state represents `false` value, it'll be reversed to `true`. The operation name is `OPERATION_REVERSE`.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

---

setTrue

```
void setTrue()
      throws UnsupportedOperationException,
             IllegalStateException,
             org.osgi.service.dal.DeviceException
```

Sets the `BooleanControl` state to `true` value. The operation name is `OPERATION_SET_TRUE`.

Throws:

`UnsupportedOperationException` - If the operation is not supported.

`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

---

setFalse

```
void setFalse()
       throws UnsupportedOperationException,
              IllegalStateException,
              org.osgi.service.dal.DeviceException
```

Sets the `BooleanControl` state to `false` value. The operation name is `OPERATION_SET_FALSE`.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

Interface BooleanSensor

org.osgi.service.dal.functions

All Superinterfaces:

org.osgi.service.dal.Function

```
public interface BooleanSensor
extends org.osgi.service.dal.Function
```

`BooleanSensor` function provides boolean sensor monitoring. It reports its state when an important event is available. The state is accessible with `getData()` getter. There are no operations.

As an example, the function is easily mappable to ZigBee Occupancy Sensing cluster and Z-Wave Binary Sensor command class. The sensor type can be:

- `Types.TYPE_LIGHT`
- `Types.TYPE_GAS`
- `Types.TYPE_SMOKE`
- `Types.TYPE_DOOR`
- `Types.TYPE_WINDOW`
- `Types.TYPE_POWER`
- `Types.TYPE_RAIN`
- `Types.TYPE_CONTACT`
- `Types.TYPE_FIRE`
- `Types.TYPE_OCCUPANCY`
- `Types.TYPE_WATER`
- `Types.TYPE_MOTION`
- other type defined in `Types`
- custom - vendor specific type

See Also:

`BooleanData`

Field Summary		Page
String	**PROPERTY_DATA** Specifies the state property name.	22

Fields inherited from interface org.osgi.service.dal.Function
SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Method Summary		Page
BooleanData	**getData**() Returns the `BooleanSensor`current state.	22

Methods inherited from interface org.osgi.service.dal.Function
getOperationMetadata, getPropertyMetadata, getServiceProperty

Field Detail

PROPERTY_DATA

```
public static final String PROPERTY_DATA = "data"
```

Specifies the state property name. The property value is accessible with `getData()` getter.

Method Detail

getData

```
BooleanData getData()
            throws UnsupportedOperationException,
                   IllegalStateException,
                   org.osgi.service.dal.DeviceException
```

Returns the `BooleanSensor`current state. It's a getter method for `PROPERTY_DATA` property.

Returns:

The `BooleanSensor` current state.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

See Also:

`BooleanData`

Interface Keypad

org.osgi.service.dal.functions

All Superinterfaces:

org.osgi.service.dal.Function

```
public interface Keypad
extends org.osgi.service.dal.Function
```

`Keypad` function provides support for keypad control. A keypad typically consists of one or more keys/buttons, which can be discerned. Different types of key presses like short and long press can typically also be detected. There is only one eventable property and no operations.

`Keypad` can enumerate all supported keys in the key property metadata, `org.osgi.service.dal.PropertyMetadata.getEnumValues(String)`. `KeypadData` event type will be `KeypadData.EVENT_TYPE_UNKNOWN` in this case.

See Also:

`KeypadData`

Field Summary		Page
String	**PROPERTY_KEY** Specifies a property name for a key from the keypad.	23

Fields inherited from interface org.osgi.service.dal.Function
SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Methods inherited from interface org.osgi.service.dal.Function
getOperationMetadata, getPropertyMetadata, getServiceProperty

Field Detail

PROPERTY_KEY

```
public static final String PROPERTY_KEY = "key"
```

Specifies a property name for a key from the keypad. The property is eventable.

See Also:

`KeypadData`

Interface Meter

org.osgi.service.dal.functions

All Superinterfaces:

org.osgi.service.dal.Function

```
public interface Meter
extends org.osgi.service.dal.Function
```

`Meter` function can measure metering information. The function provides three properties and one operation:

- PROPERTY_CURRENT
- - property accessible with getCurrent() getter;
- PROPERTY_TOTAL
- - property accessible with getTotal() getter;
- SERVICE_FLOW
- - property accessible with getTotal() getter;
- OPERATION_RESET_TOTAL
- - operation can be executed with resetTotal().

As an example, the function is easily mappable to ZigBee Simple Metering cluster and Z-Wave Meter command class. The sensor type can be:

- Types.TYPE_PRESSURE
- Types.TYPE_GAS
- Types.TYPE_POWER
- Types.TYPE_WATER
- Types.TYPE_HEAT
- Types.TYPE_COLD
- other type defined in Types
- custom - vendor specific type

See Also:

LevelData

Field Summary		Page
String	**FLOW_IN** Represents the metering consumption flow.	25
String	**FLOW_OUT** Represents the metering generation flow.	25
String	**OPERATION_RESET_TOTAL** Specifies the reset total operation name.	26
String	**PROPERTY_CURRENT** Specifies the current consumption property name.	25
String	**PROPERTY_TOTAL** Specifies the total consumption property name.	25
String	**SERVICE_FLOW** The service property value contains the metering flow.	25

Fields inherited from interface org.osgi.service.dal.Function
SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Method Summary		Page
LevelData	**getCurrent**() Returns the current metering info.	26
LevelData	**getTotal**() Returns the total metering info.	26
void	**resetTotal**() Resets the total metering info.	26

Methods inherited from interface org.osgi.service.dal.Function
getOperationMetadata, getPropertyMetadata, getServiceProperty

Field Detail

FLOW_IN

```
public static final String FLOW_IN = "in"
```

Represents the metering consumption flow. It can be used as SERVICE_FLOW property value.

FLOW_OUT

```
public static final String FLOW_OUT = "out"
```

Represents the metering generation flow. It can be used as SERVICE_FLOW property value.

SERVICE_FLOW

```
public static final String SERVICE_FLOW = "dal.meter.flow"
```

The service property value contains the metering flow. It's an optional property and available only if it's supported by the meter. The value type is `java.lang.String`. Possible property values:

- FLOW_IN
- FLOW_OUT

PROPERTY_CURRENT

```
public static final String PROPERTY_CURRENT = "current"
```

Specifies the current consumption property name. The property can be read with getCurrent() getter.

PROPERTY_TOTAL

```
public static final String PROPERTY_TOTAL = "total"
```

Specifies the total consumption property name. It has been measured since the last call of `resetTotal()` or device initial run. The property can be read with `getTotal()` getter.

OPERATION_RESET_TOTAL

```
public static final String OPERATION_RESET_TOTAL = "resetTotal"
```

Specifies the reset total operation name. The operation can be executed with `resetTotal()` method.

Method Detail

getCurrent

```
LevelData getCurrent()
              throws UnsupportedOperationException,
                     IllegalStateException,
                     org.osgi.service.dal.DeviceException
```

Returns the current metering info. It's a getter method for `PROPERTY_CURRENT` property.

Returns:
The current metering info.

Throws:
`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

See Also:
`LevelData`

---

getTotal

```
LevelData getTotal()
            throws UnsupportedOperationException,
                   IllegalStateException,
                   org.osgi.service.dal.DeviceException
```

Returns the total metering info. It's a getter method for `PROPERTY_TOTAL` property.

Returns:
The total metering info.

Throws:
`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

See Also:
`LevelData`

---

resetTotal

```
void resetTotal()
         throws UnsupportedOperationException,
                IllegalStateException,
                org.osgi.service.dal.DeviceException
```

Resets the total metering info.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

Interface MultiLevelControl

org.osgi.service.dal.functions

All Superinterfaces:

org.osgi.service.dal.Function

```
public interface MultiLevelControl
extends org.osgi.service.dal.Function
```

`MultiLevelControl` function provides multi-level control support. The function level is accessible with `getData()` getter, `setData(BigDecimal)` setter and `setData(BigDecimal, String)` setter.

As an example, the function is easily mappable to ZigBee Level Control and Z-Wave Multilevel Switch command class. The control type can be:

- `Types.TYPE_LIGHT`
- `Types.TYPE_TEMPERATURE`
- `Types.TYPE_FLOW`
- `Types.TYPE_PRESSURE`
- `Types.TYPE_HUMIDITY`
- `Types.TYPE_GAS`
- `Types.TYPE_SMOKE`
- `Types.TYPE_DOOR`
- `Types.TYPE_WINDOW`
- `Types.TYPE_LIQUID`
- `Types.TYPE_POWER`
- `Types.TYPE_NOISINESS`
- other type defined in `Types`
- custom - vendor specific type

See Also:

`LevelData`

Field Summary		Page
String	**PROPERTY_DATA** Specifies the level property name.	29

Fields inherited from interface org.osgi.service.dal.Function
SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Method Summary		Page
LevelData	**getData**() Returns `MultiLevelControl` level.	29
void	**setData**(BigDecimal level) Sets `MultiLevelControl` level to the specified value.	29
void	**setData**(BigDecimal level, String unit) Sets `MultiLevelControl` level according to the specified unit.	30

Methods inherited from interface org.osgi.service.dal.Function
getOperationMetadata, getPropertyMetadata, getServiceProperty

Field Detail

PROPERTY_DATA

```
public static final String PROPERTY_DATA = "data"
```

Specifies the level property name. The property can be read with getData() getter and can be set with setData(BigDecimal) or setData(BigDecimal, String) setters.

Method Detail

getData

```
LevelData getData()
          throws UnsupportedOperationException,
                 IllegalStateException,
                 org.osgi.service.dal.DeviceException
```

Returns MultiLevelControl level. It's a getter method for PROPERTY_DATA property.

Returns:

MultiLevelControl level.

Throws:

UnsupportedOperationException - If the operation is not supported.
IllegalStateException - If this function service object has already been unregistered.
org.osgi.service.dal.DeviceException - If an operation error is available.

See Also:

LevelData

---

setData

```
void setData(BigDecimal level)
     throws UnsupportedOperationException,
            IllegalStateException,
            org.osgi.service.dal.DeviceException,
            IllegalArgumentException
```

Sets MultiLevelControl level to the specified value. It's a setter method for PROPERTY_DATA property.

Parameters:

level - The new control level.

Throws:

UnsupportedOperationException - If the operation is not supported.
IllegalStateException - If this function service object has already been unregistered.
org.osgi.service.dal.DeviceException - If an operation error is available.
IllegalArgumentException - If there is an invalid argument.

---

setData

```
void setData(BigDecimal level,
             String unit)
      throws UnsupportedOperationException,
             IllegalStateException,
             org.osgi.service.dal.DeviceException,
             IllegalArgumentException
```

Sets `MultiLevelControl` level according to the specified unit. It's a setter method for `PROPERTY_DATA` property.

Parameters:

`level` - The new control level.

`unit` - The level unit.

Throws:

`UnsupportedOperationException` - If the operation is not supported.

`IllegalStateException` - If this function service object has already been unregistered.

`org.osgi.service.dal.DeviceException` - If an operation error is available.

`IllegalArgumentException` - If there is an invalid argument.

Interface MultiLevelSensor

org.osgi.service.dal.functions

All Superinterfaces:

org.osgi.service.dal.Function

```
public interface MultiLevelSensor
extends org.osgi.service.dal.Function
```

`MultiLevelSensor` function provides multi-level sensor monitoring. It reports its state when an important event is available. The state is accessible with `getData()` getter. There are no operations.

As an example, the function is easily mappable to ZigBee Illuminance Measurement, Temperature Measurement, Pressure Measurement, Flow Measurement and Relative Humidity Measurement cluster and Z-Wave Multilevel Sensor command class. The sensor type can be:

- `Types.TYPE_LIGHT`
- `Types.TYPE_TEMPERATURE`
- `Types.TYPE_FLOW`
- `Types.TYPE_PRESSURE`
- `Types.TYPE_HUMIDITY`
- `Types.TYPE_GAS`
- `Types.TYPE_SMOKE`
- `Types.TYPE_DOOR`
- `Types.TYPE_WINDOW`
- `Types.TYPE_LIQUID`
- `Types.TYPE_POWER`
- `Types.TYPE_NOISINESS`
- `Types.TYPE_RAIN`
- other type defined in `Types`
- custom - vendor specific type

See Also:

`LevelData`

Field Summary	Page
String **PROPERTY_DATA** Specifies the state property name.	32

Fields inherited from interface org.osgi.service.dal.Function
SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Method Summary	Page
LevelData **getData**() Returns the `MultiLevelSensor` current state.	32

Methods inherited from interface org.osgi.service.dal.Function
getOperationMetadata, getPropertyMetadata, getServiceProperty

Field Detail

PROPERTY_DATA

```
public static final String PROPERTY_DATA = "data"
```

Specifies the state property name. The property can be read with `getData()` getter.

See Also:
`LevelData`

Method Detail

getData

```
LevelData getData()
          throws UnsupportedOperationException,
                 IllegalStateException,
                 org.osgi.service.dal.DeviceException
```

Returns the `MultiLevelSensor` current state. It's a getter method for `PROPERTY_DATA` property.

Returns:
The `MultiLevelSensor` current state.

Throws:
`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

See Also:
`LevelData`

Interface Types

org.osgi.service.dal.functions

`public interface` **`Types`**

Shares common constants for all functions defined in this package. The defined function types are mapped as follow:

- `TYPE_LIGHT` - `MultiLevelControl`, `MultiLevelSensor`, `BooleanSensor` and `BooleanControl`
- `TYPE_TEMPERATURE` - `MultiLevelControl` and `MultiLevelSensor`
- `TYPE_FLOW` - `MultiLevelControl` and `MultiLevelSensor`
- `TYPE_PRESSURE` - `MultiLevelControl`, `MultiLevelSensor` and `Meter`
- `TYPE_HUMIDITY` - `MultiLevelControl` and `MultiLevelSensor`
- `TYPE_GAS` - `MultiLevelControl`, `MultiLevelSensor`, `BooleanSensor` and `Meter`
- `TYPE_SMOKE` - `MultiLevelControl`, `MultiLevelSensor` and `BooleanSensor`
- `TYPE_DOOR` - `MultiLevelControl`, `MultiLevelSensor`, `BooleanSensor` and `BooleanControl`
- `TYPE_WINDOW` - `MultiLevelControl`, `MultiLevelSensor`, `BooleanSensor` and `BooleanControl`
- `TYPE_LIQUID` - `MultiLevelControl` and `MultiLevelSensor`
- `TYPE_POWER` - `MultiLevelControl`, `MultiLevelSensor`, `BooleanSensor`, `BooleanControl` and `Meter`
- `TYPE_NOISINESS` - `MultiLevelControl` and `MultiLevelSensor`
- `TYPE_RAIN` - `MultiLevelSensor` and `BooleanSensor`
- `TYPE_CONTACT` - `BooleanSensor`
- `TYPE_FIRE` - `BooleanSensor`
- `TYPE_OCCUPANCY` - `BooleanSensor`
- `TYPE_WATER` - `BooleanSensor` and `Meter`
- `TYPE_MOTION` - `BooleanSensor`
- `TYPE_HEAT` - `Meter`
- `TYPE_COLD` - `Meter`

The mapping is not mandatory. The function can use custom defined types.

Field Summary		Page
`String`	**`TYPE_COLD`** The function type is applicable to: • `Meter` - indicates that the `Meter` measures thermal energy provided by a source. This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.	*41*
`String`	**`TYPE_CONTACT`** The function type is applicable to: • `BinarySensor` - indicates that the `BinarySensor` can detect contact.	*39*

String	**TYPE_DOOR** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the door position.	*38*
String	**TYPE_FIRE** The function type is applicable to: • `BinarySensor` - indicates that the `BinarySensor` can detect fire.	*39*
String	**TYPE_FLOW** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the flow level.	*36*
String	**TYPE_GAS** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the gas level.	*37*
String	**TYPE_HEAT** The function type is applicable to: • `Meter` - indicates that the `Meter` measures thermal energy provided by a source. This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.	*40*
String	**TYPE_HUMIDITY** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the humidity level.	*37*
String	**TYPE_LIGHT** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control light devices.	*36*
String	**TYPE_LIQUID** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the liquid level.	*38*
String	**TYPE_MOTION** The function type is applicable to: • `BinarySensor` - indicates that the `BinarySensor` can detect motion.	*40*

String	**TYPE_NOISINESS** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the noise level.	39
String	**TYPE_OCCUPANCY** The function type is applicable to: • `BinarySensor` - indicates that the `BinarySensor` can detect presence.	40
String	**TYPE_POWER** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the power level.	38
String	**TYPE_PRESSURE** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the pressure level.	36
String	**TYPE_RAIN** The function type is applicable to: • `MultiLevelSensor` - indicates that the `MultiLevelSensor` can monitor the rain rate.	39
String	**TYPE_SMOKE** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the smoke level.	37
String	**TYPE_TEMPERATURE** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control temperature devices.	36
String	**TYPE_WATER** The function type is applicable to: • `BinarySensor` - indicates that the `BinarySensor` can detect water leak.	40
String	**TYPE_WINDOW** The function type is applicable to: • `MultiLevelControl` - indicates that the `MultiLevelControl` can control the window position.	38

Field Detail

TYPE_LIGHT

```
public static final String TYPE_LIGHT = "light"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control light devices. Usually, such devices are called dimmable. `MultiLevelControl` minimum value can switch off the device and `MultiLevelControl` maximum value can increase the device light to the maximum possible value.
- `MultiLevelSensor` - indicates that the sensor can monitor the light level.
- `BinarySensor` - indicates that the `BinarySensor` can detected light. `true` state means that there is light. `false` state means that there is no light.
- `BinaryControl` - indicates that there is a light device control. `true` state means that the light device will be turned on. `false` state means that the light device will be turned off.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_TEMPERATURE

```
public static final String TYPE_TEMPERATURE = "temperature"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control temperature devices. For example, such device can be thermostat. `MultiLevelControl` minimum value is the lowest supported temperature. `MultiLevelControl` maximum value is the highest supported temperature.
- `MultiLevelSensor` - indicates that the sensor can monitor the temperature.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_FLOW

```
public static final String TYPE_FLOW = "flow"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the flow level. `MultiLevelControl` minimum value is the minimum supported flow level. `MultiLevelControl` maximum value is the maximum supported flow level.
- `MultiLevelSensor` - indicates that the sensor can monitor the flow level.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_PRESSURE

```
public static final String TYPE_PRESSURE = "pressure"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the pressure level. `MultiLevelControl` minimum value is the lowest supported pressure level. `MultiLevelControl` maximum value is the highest supported pressure level.
- `MultiLevelSensor` - indicates that the sensor can monitor the pressure level.
- `Meter` - Indicates that the `Meter` measures pressure.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_HUMIDITY

```
public static final String TYPE_HUMIDITY = "humidity"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the humidity level. It's typical functionality for HVAC (heating, ventilation, and air conditioning) devices. `MultiLevelControl` minimum value is the lowest supported humidity level. `MultiLevelControl` maximum value is the highest supported humidity level.
- `MultiLevelSensor` - indicates that the sensor can monitor the humidity level.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_GAS

```
public static final String TYPE_GAS = "gas"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the gas level. `MultiLevelControl` minimum value is the lowest supported gas level. `MultiLevelControl` maximum value is the highest supported gas level.
- `MultiLevelSensor` - indicates that the sensor can monitor the gas level.
- `BinarySensor` - indicates that the `BinarySensor` supports gas detection. `true` state means there is gas. `false` state means that there is no gas.
- `Meter` - indicates that the `Meter` measures the gas consumption.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_SMOKE

```
public static final String TYPE_SMOKE = "smoke"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the smoke level. `MultiLevelControl` minimum value is the lowest supported smoke level. `MultiLevelControl` maximum value is the highest supported smoke level.
- `MultiLevelSensor` - indicates that the sensor can monitor the smoke level.
- `BinarySensor` - indicates that the `BinarySensor` can detect smoke. `true` state means that there is smoke. `false` state means that there is no rain.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_DOOR

```
public static final String TYPE_DOOR = "door"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the door position. `MultiLevelControl` minimum value can completely close the door. `MultiLevelControl` maximum value can open the door to the maximum allowed position.
- `MultiLevelSensor` - indicates that the sensor can monitor the door position.
- `BinarySensor` - indicates that the `BinarySensor` can detect the door state. `true` state means that the door is opened. `false` state means that the door is closed.
- `BinaryControl` - indicates that there is a door position control. `true` state means that the door will be opened. `false` state means that the the door will be closed.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_WINDOW

```
public static final String TYPE_WINDOW = "window"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the window position. `MultiLevelControl` minimum value can completely close the window. `MultiLevelControl` maximum value can open the window to the maximum allowed position.
- `MultiLevelSensor` - indicates that the sensor can monitor the window position.
- `BinarySensor` - indicates that the `BinarySensor` can window state. `true` state means that the window is opened. `false` state means that the window is closed.
- `BinaryControl` - indicates that there is a window position control. `true` state means that the window will be opened. `false` state means that the the window will be closed.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_LIQUID

```
public static final String TYPE_LIQUID = "liquid"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the liquid level. `MultiLevelControl` minimum value is the lowest supported liquid level. `MultiLevelControl` maximum value is the highest supported liquid level.
- `MultiLevelSensor` - indicates that the sensor can monitor the liquid level.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_POWER

```
public static final String TYPE_POWER = "power"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the power level. `MultiLevelControl` minimum value is the lowest supported power level. `MultiLevelControl` maximum value is the highest supported power level.
- `MultiLevelSensor` - indicates that the sensor can monitor the power level.
- `BinarySensor` - indicates that the `BinarySensor` can detect motion. `true` state means that there is power restore. `false` state means that there is power cut.
- `BinaryControl` - indicates that there is electricity control. `true` state means that the power will be restored. `false` state means that the power will be cut.
- `Meter` - indicates that the `Meter` measures the power consumption.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_NOISINESS

```
public static final String TYPE_NOISINESS = "noisiness"
```

The function type is applicable to:

- `MultiLevelControl` - indicates that the `MultiLevelControl` can control the noise level. `MultiLevelControl` minimum value is the lowest supported noise level. `MultiLevelControl` maximum value is the highest supported noise level.
- `MultiLevelSensor` - indicates that the sensor can monitor the noise level.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_RAIN

```
public static final String TYPE_RAIN = "rain"
```

The function type is applicable to:

- `MultiLevelSensor` - indicates that the `MultiLevelSensor` can monitor the rain rate. It's not applicable to `MultiLevelControl`.
- `BinarySensor` - indicates that the `BinarySensor` can detect rain. `true` state means that there is rain. `false` state means that there is no rain.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_CONTACT

```
public static final String TYPE_CONTACT = "contact"
```

The function type is applicable to:

- `BinarySensor` - indicates that the `BinarySensor` can detect contact. `true` state means that there is contact. `false` state means that there is no contact.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_FIRE

```
public static final String TYPE_FIRE = "fire"
```

The function type is applicable to:

- `BinarySensor` - indicates that the `BinarySensor` can detect fire. `true` state means that there is fire. `false` state means that there is no fire.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_OCCUPANCY

```
public static final String TYPE_OCCUPANCY = "occupancy"
```

The function type is applicable to:

- `BinarySensor` - indicates that the `BinarySensor` can detect presence. `true` state means that someone is detected. `false` state means that nobody is detected.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_WATER

```
public static final String TYPE_WATER = "water"
```

The function type is applicable to:

- `BinarySensor` - indicates that the `BinarySensor` can detect water leak. `true` state means that there is water leak. `false` state means that there is no water leak.
- `Meter` - indicates that the `Meter` measures water consumption.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_MOTION

```
public static final String TYPE_MOTION = "motion"
```

The function type is applicable to:

- `BinarySensor` - indicates that the `BinarySensor` can detect motion. `true` state means that there is motion detection. `false` state means that there is no motion detection.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_HEAT

```
public static final String TYPE_HEAT = "heat"
```

The function type is applicable to:

- `Meter` - indicates that the `Meter` measures thermal energy provided by a source.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

TYPE_COLD

```
public static final String TYPE_COLD = "cold"
```

The function type is applicable to:

- `Meter` - indicates that the `Meter` measures thermal energy provided by a source.

This type can be specified as a value of `org.osgi.service.dal.Function.SERVICE_TYPE`.

Interface WakeUp

org.osgi.service.dal.functions

All Superinterfaces:

org.osgi.service.dal.Function

```
public interface WakeUp
extends org.osgi.service.dal.Function
```

`WakeUp` function provides device awake monitoring and management. It's especially applicable to battery-operated devices. Such device can notify the system that it's awake and can receive commands with an event to property `PROPERTY_AWAKE`.

The device can periodically wake up for commands. The interval can be managed with `PROPERTY_WAKE_UP_INTERVAL` property.

The application can minimize the power consumption with `sleep()` operation. As a result, the device will sleep and will not receive commands to the next awake.

See Also:

`LevelData`, `BooleanData`

Field Summary		Page
String	**OPERATION_SLEEP** Specifies the sleep operation name.	43
String	**PROPERTY_AWAKE** Specifies the awake property name.	43
String	**PROPERTY_WAKE_UP_INTERVAL** Specifies the wake up interval.	43

Fields inherited from interface org.osgi.service.dal.Function
SERVICE_DESCRIPTION, SERVICE_DEVICE_UID, SERVICE_OPERATION_NAMES, SERVICE_PROPERTY_NAMES, SERVICE_REFERENCE_UIDS, SERVICE_TYPE, SERVICE_UID, SERVICE_VERSION

Method Summary		Page
LevelData	**getWakeUpInterval**() Returns the current wake up interval.	43
void	**setWakeUpInterval**(BigDecimal interval) Sets wake up interval according to the default unit.	44
void	**setWakeUpInterval**(BigDecimal interval, String unit) Sets wake up interval according to the specified unit.	44
void	**sleep**() The device is forced to sleep to minimize the power consumption.	44

Methods inherited from interface org.osgi.service.dal.Function
getOperationMetadata, getPropertyMetadata, getServiceProperty

Field Detail

PROPERTY_AWAKE

```
public static final String PROPERTY_AWAKE = "awake"
```

Specifies the awake property name. The property access type can be `org.osgi.service.dal.PropertyMetadata.PROPERTY_ACCESS_EVENTABLE`. If the device is awake, it will trigger a property event.

The property value type is `BooleanData`. The boolean data is always `true`. It marks that the device is awake.

PROPERTY_WAKE_UP_INTERVAL

```
public static final String PROPERTY_WAKE_UP_INTERVAL = "wakeUpInterval"
```

Specifies the wake up interval. The device can periodically wake up and receive commands. That interval is managed by this property. The current property value is available with `getWakeUpInterval()` and can be modified with `setWakeUpInterval(BigDecimal)` and `setWakeUpInterval(BigDecimal, String)`.

OPERATION_SLEEP

```
public static final String OPERATION_SLEEP = "sleep"
```

Specifies the sleep operation name. The operation can be executed with `sleep()` method.

Method Detail

getWakeUpInterval

```
LevelData getWakeUpInterval()
                      throws UnsupportedOperationException,
                             IllegalStateException,
                             org.osgi.service.dal.DeviceException
```

Returns the current wake up interval. It's a getter method for `PROPERTY_WAKE_UP_INTERVAL` property. The device can periodically wake up and receive command based on this interval.

The interval can be measured in different units like hours, minutes, seconds etc. The unit is specified in `LevelData` instance.

Returns:

The current wake up interval.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

See Also:

`LevelData`

---

setWakeUpInterval

```
void setWakeUpInterval(BigDecimal interval)
                throws UnsupportedOperationException,
                       IllegalStateException,
                       org.osgi.service.dal.DeviceException,
                       IllegalArgumentException
```

Sets wake up interval according to the default unit. It's a setter method for PROPERTY_WAKE_UP_INTERVAL property. The device can periodically wake up and receive command based on this interval.

Parameters:

`interval` - The new wake up interval.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.
`IllegalArgumentException` - If there is an invalid argument.

---

setWakeUpInterval

```
void setWakeUpInterval(BigDecimal interval,
                       String unit)
                throws UnsupportedOperationException,
                       IllegalStateException,
                       org.osgi.service.dal.DeviceException,
                       IllegalArgumentException
```

Sets wake up interval according to the specified unit. It's a setter method for PROPERTY_WAKE_UP_INTERVAL property. The device can periodically wake up and receive command based on this interval.

Parameters:

`interval` - The new wake up interval.
`unit` - The interval unit.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.
`IllegalArgumentException` - If there is an invalid argument.

---

sleep

```
void sleep()
    throws UnsupportedOperationException,
           IllegalStateException,
           org.osgi.service.dal.DeviceException
```

The device is forced to sleep to minimize the power consumption.

Throws:

`UnsupportedOperationException` - If the operation is not supported.
`IllegalStateException` - If this function service object has already been unregistered.
`org.osgi.service.dal.DeviceException` - If an operation error is available.

Package org.osgi.service.dal.functions.data

Function Data 1.0.

See:

[Description]

Class Summary		Page
AlarmData	Function alarm data.	46
BooleanData	Function boolean data wrapper.	53
KeypadData	Represents a keypad event data that is collected when a change with some key from device keypad has occurred.	57
LevelData	Function level data wrapper.	63

Package org.osgi.service.dal.functions.data Description

Function Data 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.functions.data; version="[1.0,2.0)"
```

Example import for providers implementing the API in this package:

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

Class AlarmData

org.osgi.service.dal.functions.data

```
java.lang.Object
  └org.osgi.service.dal.FunctionData
      └org.osgi.service.dal.functions.data.AlarmData
```

All Implemented Interfaces:

Comparable

```
public class AlarmData
extends org.osgi.service.dal.FunctionData
```

Function alarm data. It cares about the alarm type, severity, timestamp and additional metadata. It doesn't support unit. The alarm type is mapped to `FunctionData` value.

See Also:

Alarm, `org.osgi.service.dal.FunctionData`

Field Summary		Page
static String	**FIELD_SEVERITY** Represents the severity field name.	47
static String	**FIELD_TYPE** Represents the type field name.	47
int	**severity** Represents the alarm severity.	49
static int	**SEVERITY_HIGH** The severity rating indicates that there is an alarm with high priority.	49
static int	**SEVERITY_LOW** The severity rating indicates that there is an alarm with lowest priority.	49
static int	**SEVERITY_MEDIUM** The severity rating indicates that there is an alarm with medium priority.	49
static int	**SEVERITY_NONE** The severity constant indicates that there is no severity rating for this alarm.	49
static int	**SEVERITY_URGENT** The severity rating indicates that there an urgent alarm.	49
int	**type** Represents the alarm type.	50
static int	**TYPE_COLD** The alarm type indicates that temperature is too low.	48
static int	**TYPE_GAS_CO** The alarm type indicates that carbon monoxide is detected.	48
static int	**TYPE_GAS_CO2** The alarm type indicates that carbon dioxide is detected.	48
static int	**TYPE_HEAT** The alarm type indicates that temperature is too high.	48
static int	**TYPE_HW_FAIL** The alarm type indicates that there is hardware failure.	49

static int	**TYPE_POWER_FAIL** The alarm type indicates a power cut.	48
static int	**TYPE_SMOKE** The alarm type indicates that smoke is detected.	48
static int	**TYPE_SW_FAIL** The alarm type indicates that there is software failure.	49
static int	**TYPE_WATER** The alarm type indicates that water leak is detected.	48

Fields inherited from class org.osgi.service.dal.FunctionData
FIELD_METADATA, FIELD_TIMESTAMP, META_INFO_DESCRIPTION, metadata, timestamp

Constructor Summary	Page
AlarmData(Map fields) Constructs new AlarmData instance with the specified field values.	50
AlarmData(long timestamp, Map metadata, int severity, int type) Constructs new AlarmData instance with the specified arguments.	50

Method Summary		Page
int	**compareTo**(Object o) Compares this AlarmData instance with the given argument.	52
boolean	**equals**(Object other) Two AlarmData instances are equal if they contain equal metadata, timestamp, type and severity.	51
int	**getSeverity**() Returns the alarm severity.	51
int	**getType**() Returns the alarm type.	50
int	**hashCode**() Returns the hash code for this AlarmData object.	51

Methods inherited from class org.osgi.service.dal.FunctionData
getMetadata, getTimestamp

Field Detail

FIELD_SEVERITY

```
public static final String FIELD_SEVERITY = "severity"
```

Represents the severity field name. The field value is available with severity and getSeverity(). The field type is int. The constant can be used as a key to AlarmData(Map).

FIELD_TYPE

```
public static final String FIELD_TYPE = "type"
```

Represents the type field name. The field value is available with type and getType(). The field type is int. The constant can be used as a key to AlarmData(Map).

TYPE_SMOKE

```
public static final int TYPE_SMOKE = 1
```

The alarm type indicates that smoke is detected.

TYPE_HEAT

```
public static final int TYPE_HEAT = 2
```

The alarm type indicates that temperature is too high.

TYPE_COLD

```
public static final int TYPE_COLD = 3
```

The alarm type indicates that temperature is too low.

TYPE_GAS_CO2

```
public static final int TYPE_GAS_CO2 = 4
```

The alarm type indicates that carbon dioxide is detected.

TYPE_GAS_CO

```
public static final int TYPE_GAS_CO = 5
```

The alarm type indicates that carbon monoxide is detected.

TYPE_WATER

```
public static final int TYPE_WATER = 6
```

The alarm type indicates that water leak is detected.

TYPE_POWER_FAIL

```
public static final int TYPE_POWER_FAIL = 7
```

The alarm type indicates a power cut.

TYPE_HW_FAIL

```
public static final int TYPE_HW_FAIL = 8
```

The alarm type indicates that there is hardware failure.

TYPE_SW_FAIL

```
public static final int TYPE_SW_FAIL = 9
```

The alarm type indicates that there is software failure.

SEVERITY_NONE

```
public static final int SEVERITY_NONE = 0
```

The severity constant indicates that there is no severity rating for this alarm.

SEVERITY_LOW

```
public static final int SEVERITY_LOW = 1
```

The severity rating indicates that there is an alarm with lowest priority.

SEVERITY_MEDIUM

```
public static final int SEVERITY_MEDIUM = 2
```

The severity rating indicates that there is an alarm with medium priority. The severity priority is higher than SEVERITY_LOW and lower than SEVERITY_HIGH.

SEVERITY_HIGH

```
public static final int SEVERITY_HIGH = 3
```

The severity rating indicates that there is an alarm with high priority. The severity priority is higher than SEVERITY_MEDIUM and lower than SEVERITY_URGENT.

SEVERITY_URGENT

```
public static final int SEVERITY_URGENT = 4
```

The severity rating indicates that there an urgent alarm. That severity has highest priority.

---

severity

```
public final int severity
```

Represents the alarm severity. The field is accessible with `getSeverity()` getter. The vendor can define own alarm severity ratings with negative values.

---

type

```
public final int type
```

Represents the alarm type. The field is accessible with `getType()` getter. The vendor can define own alarm types with negative values.

Constructor Detail

AlarmData

```
public AlarmData(Map fields)
```

Constructs new `AlarmData` 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: {"severity"=Integer(1)...}. That map will initialize the `FIELD_SEVERITY` field with 1. If severity is missing, `SEVERITY_NONE` is used.

`FIELD_SEVERITY` field value type must be `Integer`. `FIELD_TYPE` field value type must be `Integer`.

Parameters:

`fields` - Contains the new `AlarmData` instance field values.

Throws:

`ClassCastException` - If the field value types are not expected.
`IllegalArgumentException` - If the alarm type is missing.
`NullPointerException` - If the fields map is `null`.

AlarmData

```
public AlarmData(long timestamp,
                 Map metadata,
                 int severity,
                 int type)
```

Constructs new `AlarmData` instance with the specified arguments.

Parameters:

`timestamp` - The alarm data timestamp.
`metadata` - The alarm data metadata.
`severity` - The alarm data severity.
`type` - The alarm data type.

Method Detail

getType

```
public int getType()
```

Returns the alarm type. The type can be one of the predefined:

- TYPE_SMOKE
- TYPE_HEAT
- TYPE_COLD
- TYPE_GAS_CO
- TYPE_GAS_CO2
- TYPE_WATER
- TYPE_POWER_FAIL
- TYPE_HW_FAIL
- TYPE_SW_FAIL

The vendor can define own alarm types with negative values.

Returns:
The alarm type.

---

getSeverity

```
public int getSeverity()
```

Returns the alarm severity.

Returns:
The alarm severity.

---

equals

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

Two `AlarmData` instances are equal if they contain equal metadata, timestamp, type and severity.

Overrides:
`equals` in class `org.osgi.service.dal.FunctionData`
Parameters:
`other` - The object to compare this data.
Returns:
`true` if this object is equivalent to the specified one.
See Also:
`org.osgi.service.dal.FunctionData.equals(java.lang.Object)`

---

hashCode

```
public int hashCode()
```

Returns the hash code for this `AlarmData` object. The hash code is a sum of `org.osgi.service.dal.FunctionData.hashCode()`, the alarm severity and the alarm type.

Overrides:
`hashCode` in class `org.osgi.service.dal.FunctionData`
Returns:
The hash code of this `AlarmData` object.
See Also:
`org.osgi.service.dal.FunctionData.hashCode()`

compareTo

```
public int compareTo(Object o)
```

Compares this `AlarmData` instance with the given argument. The argument can be:

- `AlarmData` - the method returns `-1` if metadata, timestamp, type or severity are not equivalent. 0 if all fields are equivalent. 1 if all fields are equivalent and this instance severity is greater than the severity of the specified argument.
- `Map` - the map must be built according the rules of `AlarmData(Map)`. Metadata, timestamp, type and severity are compared according `AlarmData` argument rules.

Specified by:

`compareTo` in interface `Comparable`

Parameters:

`o` - An argument to be compared.

Returns:

-1, 0 or 1 depending on the comparison rules.

Throws:

`ClassCastException` - If the method is called with `Map` and the field value types are not expected.
`IllegalArgumentException` - If the method is called with `Map` and the alarm type is missing.
`NullPointerException` - If the argument is `null`.

See Also:

`Comparable.compareTo(java.lang.Object)`

Class BooleanData

org.osgi.service.dal.functions.data

```
java.lang.Object
  └org.osgi.service.dal.FunctionData
      └org.osgi.service.dal.functions.data.BooleanData
```

All Implemented Interfaces:

Comparable

```
public class BooleanData
extends org.osgi.service.dal.FunctionData
```

Function boolean data wrapper. It can contain a boolean value, timestamp and additional metadata. It doesn't support measurement unit.

See Also:

BooleanControl, BooleanSensor, org.osgi.service.dal.FunctionData

---

Field Summary	Page
static String **FIELD_VALUE** Represents the value field name.	54
boolean **value** Represents the boolean value.	54

Fields inherited from class org.osgi.service.dal.FunctionData
FIELD_METADATA, FIELD_TIMESTAMP, META_INFO_DESCRIPTION, metadata, timestamp

Constructor Summary	Page
BooleanData(Map fields) Constructs new BooleanData instance with the specified field values.	54
BooleanData(long timestamp, Map metadata, boolean value) Constructs new BooleanData instance with the specified arguments.	54

Method Summary	Page
int **compareTo**(Object o) Compares this BooleanData instance with the given argument.	55
boolean **equals**(Object other) Two BooleanData instances are equal if they contain equal metadata, timestamp and boolean value.	55
boolean **getValue**() Returns BooleanData value.	54
int **hashCode**() Returns the hash code for this BooleanData object.	55

Methods inherited from class org.osgi.service.dal.FunctionData
getMetadata, getTimestamp

Field Detail

FIELD_VALUE

```
public static final String FIELD_VALUE = "value"
```

Represents the value field name. The field value is available with `value` and `getValue()`. The field type is `boolean`. The constant can be used as a key to `BooleanData(Map)`.

---

value

```
public final boolean value
```

Represents the boolean value. The field is accessible with `getValue()` getter.

Constructor Detail

BooleanData

```
public BooleanData(Map fields)
```

Constructs new `BooleanData` 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: {"value"=Boolean(true)...}. That map will initialize the `FIELD_VALUE` field with `true`.

`FIELD_VALUE` field value type must be `Boolean`.

Parameters:

`fields` - Contains the new `BooleanData` instance field values.

Throws:

`ClassCastException` - If the field value types are not expected.
`IllegalArgumentException` - If the value is missing.
`NullPointerException` - If the fields map is `null`.

BooleanData

```
public BooleanData(long timestamp,
                   Map metadata,
                   boolean value)
```

Constructs new `BooleanData` instance with the specified arguments.

Parameters:

`timestamp` - The boolean data timestamp.
`metadata` - The boolean data metadata.
`value` - The boolean value.

Method Detail

getValue

```
public boolean getValue()
```

Returns `BooleanData` value.

Returns:

`BooleanData` value.

---

equals

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

Two `BooleanData` instances are equal if they contain equal metadata, timestamp and boolean value.

Overrides:

`equals` in class `org.osgi.service.dal.FunctionData`

Parameters:

`other` - The object to compare this data.

Returns:

`true` if this object is equivalent to the specified one.

See Also:

`org.osgi.service.dal.FunctionData.equals(java.lang.Object)`

---

hashCode

```
public int hashCode()
```

Returns the hash code for this `BooleanData` object. The hash code is a sum of `org.osgi.service.dal.FunctionData.hashCode()` and `Boolean.hashCode()`, where `Boolean.hashCode()` represents the boolean value hash code.

Overrides:

`hashCode` in class `org.osgi.service.dal.FunctionData`

Returns:

The hash code of this `BooleanData` object.

See Also:

`org.osgi.service.dal.FunctionData.hashCode()`

---

compareTo

```
public int compareTo(Object o)
```

Compares this `BooleanData` instance with the given argument. The argument can be:

- `Boolean` - the method returns `0` if this instance contains equivalent boolean value. `-1` if this instance contains `false` and the argument is `true`. `1` if this instance contains `true` and the argument is `false`.
- `BooleanData` - the method returns `-1` if metadata or timestamp are not equivalent. Otherwise, the boolean value is compared with the same rules as `Boolean` argument.
- `Map` - the map must be built according the rules of `BooleanData(Map)`. Metadata, timestamp and value are compared according `BooleanData` and `Boolean` argument rules.

Specified by:

`compareTo` in interface `Comparable`

Parameters:

`o` - An argument to be compared.

Returns:

-1, 0 or 1 depending on the comparison rules.

Throws:

`ClassCastException` - If the method is called with `Map` and field value types are not expected.
`IllegalArgumentException` - If the method is called with `Map` and the value is missing.
`NullPointerException` - If the argument is `null`.

See Also:

`Comparable.compareTo(java.lang.Object)`

Class KeypadData

org.osgi.service.dal.functions.data

```
java.lang.Object
  └org.osgi.service.dal.FunctionData
      └org.osgi.service.dal.functions.data.KeypadData
```

All Implemented Interfaces:

Comparable

```
public class KeypadData
extends org.osgi.service.dal.FunctionData
```

Represents a keypad event data that is collected when a change with some key from device keypad has occurred. The key code is mapped to `FunctionData` value.

See Also:

Keypad, org.osgi.service.dal.FunctionData

Field Summary		Page
static int	**EVENT_TYPE_PRESSED** Represents a keypad event type for a key pressed.	59
static int	**EVENT_TYPE_PRESSED_DOUBLE** Represents a keypad event type for a double key pressed.	59
static int	**EVENT_TYPE_PRESSED_DOUBLE_LONG** Represents a keypad event type for a double and long key pressed.	59
static int	**EVENT_TYPE_PRESSED_LONG** Represents a keypad event type for a long key pressed.	59
static int	**EVENT_TYPE_RELEASED** Represents a keypad event type for a key released.	59
static int	**EVENT_TYPE_UNKNOWN** Represents an unknown keypad event type.	59
int	**eventType** Represents the keypad event type.	59
static String	**FIELD_EVENT_TYPE** Represents the event type field name.	58
static String	**FIELD_KEY_CODE** Represents the key code field name.	58
static String	**FIELD_KEY_NAME** Represents the key name field name.	58
int	**keyCode** Represents the key code.	60
String	**keyName** Represents the key name, if it's available.	59

Fields inherited from class org.osgi.service.dal.FunctionData
FIELD_METADATA, FIELD_TIMESTAMP, META_INFO_DESCRIPTION, metadata, timestamp

Constructor Summary	Page
KeypadData(Map fields) Constructs new KeypadData instance with the specified field values.	60
KeypadData(long timestamp, Map metadata, int eventType, int keyCode, String keyName) Constructs new KeypadData instance with the specified arguments.	60

Method Summary		Page
int	**compareTo**(Object o) Compares this KeypadData instance with the given argument.	62
boolean	**equals**(Object other) Two KeypadData instances are equal if they contain equal metadata, timestamp, event type, key code and key name.	61
int	**getEventType**() Returns the event type.	60
int	**getKeyCode**() The code of the key.	61
String	**getKeyName**() Represents a human readable name of the corresponding key code.	61
int	**hashCode**() Returns the hash code for this KeypadData object.	61

Methods inherited from class org.osgi.service.dal.FunctionData
getMetadata, getTimestamp

Field Detail

FIELD_KEY_NAME

```
public static final String FIELD_KEY_NAME = "keyName"
```

Represents the key name field name. The field value is available with keyName and getKeyName(). The field type is String. The constant can be used as a key to KeypadData(Map).

FIELD_EVENT_TYPE

```
public static final String FIELD_EVENT_TYPE = "eventType"
```

Represents the event type field name. The field value is available with eventType and getEventType(). The field type is int. The constant can be used as a key to KeypadData(Map).

FIELD_KEY_CODE

```
public static final String FIELD_KEY_CODE = "keyCode"
```

Represents the key code field name. The field value is available with keyCode and getKeyCode(). The field type is int. The constant can be used as a key to KeypadData(Map).

EVENT_TYPE_UNKNOWN

```
public static final int EVENT_TYPE_UNKNOWN = 0
```

Represents an unknown keypad event type.

EVENT_TYPE_PRESSED

```
public static final int EVENT_TYPE_PRESSED = 1
```

Represents a keypad event type for a key pressed.

EVENT_TYPE_PRESSED_LONG

```
public static final int EVENT_TYPE_PRESSED_LONG = 2
```

Represents a keypad event type for a long key pressed.

EVENT_TYPE_PRESSED_DOUBLE

```
public static final int EVENT_TYPE_PRESSED_DOUBLE = 3
```

Represents a keypad event type for a double key pressed.

EVENT_TYPE_PRESSED_DOUBLE_LONG

```
public static final int EVENT_TYPE_PRESSED_DOUBLE_LONG = 4
```

Represents a keypad event type for a double and long key pressed.

EVENT_TYPE_RELEASED

```
public static final int EVENT_TYPE_RELEASED = 5
```

Represents a keypad event type for a key released.

---

eventType

```
public final int eventType
```

Represents the keypad event type. The vendor can define own event types with negative values. The field is accessible with `getEventType()` getter.

---

keyName

```
public final String keyName
```

Represents the key name, if it's available. The field is accessible with `getKeyName()` getter.

keyCode

```
public final int keyCode
```

Represents the key code. This field is mandatory and it holds the semantics(meaning) of the key. The field is accessible with `getKeyCode()` getter.

Constructor Detail

KeypadData

```
public KeypadData(Map fields)
```

Constructs new `KeypadData` 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: {"eventType"=Integer(1)...}. That map will initialize the `FIELD_EVENT_TYPE` field with 1.

`FIELD_EVENT_TYPE` field value type must be `Integer`. `FIELD_KEY_CODE` field value type must be `Integer`. `FIELD_KEY_NAME` field value type must be `String`.

Parameters:

`fields` - Contains the new `KeypadData` instance field values.

Throws:

`ClassCastException` - If the field value types are not expected.
`IllegalArgumentException` - If the event type or key code is missing.
`NullPointerException` - If the fields map is `null`.

KeypadData

```
public KeypadData(long timestamp,
                  Map metadata,
                  int eventType,
                  int keyCode,
                  String keyName)
```

Constructs new `KeypadData` instance with the specified arguments.

Parameters:

`timestamp` - The data timestamp.
`metadata` - The data metadata.
`eventType` - The data event type.
`keyCode` - The data key code.
`keyName` - The data key name.

Method Detail

getEventType

```
public int getEventType()
```

Returns the event type. The vendor can define own event types with negative values.

Returns:
The event type.

---

getKeyCode

```
public int getKeyCode()
```

The code of the key. This field is mandatory and it holds the semantics(meaning) of the key.

Returns:
The key code.

---

getKeyName

```
public String getKeyName()
```

Represents a human readable name of the corresponding key code. This field is optional and sometimes it could be missed(might be `null`).

Returns:
A string with the name of the key or `null` if not specified.

---

equals

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

Two `KeypadData` instances are equal if they contain equal metadata, timestamp, event type, key code and key name.

Overrides:
`equals` in class `org.osgi.service.dal.FunctionData`
Parameters:
`other` - The object to compare this data.
Returns:
`true` if this object is equivalent to the specified one.
See Also:
`org.osgi.service.dal.FunctionData.equals(java.lang.Object)`

---

hashCode

```
public int hashCode()
```

Returns the hash code for this `KeypadData` object. The hash code is a sum of `org.osgi.service.dal.FunctionData.hashCode()`, `String.hashCode()`, event type and key code, where `String.hashCode()` represents the key name hash code if available.

Overrides:
`hashCode` in class `org.osgi.service.dal.FunctionData`
Returns:
The hash code of this `LevelData` object.

See Also:

`org.osgi.service.dal.FunctionData.hashCode()`

---

compareTo

```
public int compareTo(Object o)
```

Compares this `KeypadData` instance with the given argument. The argument can be:

- `KeypadData` - the method returns `-1` if metadata, timestamp, event type, key code or key name are not equivalent. 0 if all fields are equivalent.
- `Map` - the map must be built according the rules of `KeypadData(Map)`. Metadata, timestamp, event type, key code and key name are compared according `KeypadData` argument rules.

Specified by:

`compareTo` in interface `Comparable`

Parameters:

`o` - An argument to be compared.

Returns:

-1 or 0 depending on the comparison rules.

Throws:

`ClassCastException` - If the method is called with `Map` and the field value types are not expected.

`IllegalArgumentException` - If the method is called with `Map` and the event type or key code is missing.

`NullPointerException` - If the argument is `null`.

See Also:

`Comparable.compareTo(java.lang.Object)`

Class LevelData

org.osgi.service.dal.functions.data

```
java.lang.Object
  └org.osgi.service.dal.FunctionData
      └org.osgi.service.dal.functions.data.LevelData
```

All Implemented Interfaces:

Comparable

```
public class LevelData
extends org.osgi.service.dal.FunctionData
```

Function level data wrapper. It supports all properties defined in `FunctionData`.

See Also:

`MultiLevelControl`, `MultiLevelSensor`, `Meter`, `org.osgi.service.dal.FunctionData`

Field Summary		Page
static String	**FIELD_LEVEL** Represents the level field name.	64
static String	**FIELD_UNIT** Represents the unit field name.	64
BigDecimal	**level** Represents the current level.	64
String	**unit** Represent the unit as it's defined in `org.osgi.service.dal.PropertyMetadata.UNITS`.	64

Fields inherited from class org.osgi.service.dal.FunctionData
`FIELD_METADATA, FIELD_TIMESTAMP, META_INFO_DESCRIPTION, metadata, timestamp`

Constructor Summary	Page
LevelData`(Map fields)` Constructs new `LevelData` instance with the specified field values.	64
LevelData`(long timestamp, Map metadata, String unit, BigDecimal level)` Constructs new `LevelData` instance with the specified arguments.	65

Method Summary		Page
int	**compareTo**`(Object o)` Compares this `LevelData` instance with the given argument.	66
boolean	**equals**`(Object other)` Two `LevelData` instances are equal if they contain equal metadata, timestamp, unit and level.	65
BigDecimal	**getLevel**`()` Returns `LevelData` value.	65
String	**getUnit**`()` Returns `LevelData` unit as it's specified in `org.osgi.service.dal.PropertyMetadata.UNITS` or `null` if the unit is missing.	65

int	**hashCode**() Returns the hash code for this LevelData object.	66

Methods inherited from class org.osgi.service.dal.FunctionData
getMetadata, getTimestamp

Field Detail

FIELD_LEVEL

```
public static final String FIELD_LEVEL = "level"
```

Represents the level field name. The field value is available with level and getLevel(). The field type is BigDecimal. The constant can be used as a key to LevelData(Map).

FIELD_UNIT

```
public static final String FIELD_UNIT = "unit"
```

Represents the unit field name. The field value is available with unit and getUnit(). The field type is String. The constant can be used as a key to LevelData(Map).

---

unit

```
public final String unit
```

Represent the unit as it's defined in org.osgi.service.dal.PropertyMetadata.UNITS. The field is optional. The field is accessible with getUnit() getter.

---

level

```
public final BigDecimal level
```

Represents the current level. It's mandatory field. The field is accessible with getLevel() getter.

Constructor Detail

LevelData

```
public LevelData(Map fields)
```

Constructs new LevelData 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: {"level"=BigDecimal(1)...}. That map will initialize the FIELD_LEVEL field with 1.

FIELD_UNIT field value type must be String. FIELD_LEVEL field value type must be BigDecimal.

Parameters:

fields - Contains the new LevelData instance field values.

Throws:

`ClassCastException` - If the field value types are not expected.
`IllegalArgumentException` - If the level is missing.
`NullPointerException` - If the fields map is `null`.

LevelData

```
public LevelData(long timestamp,
                 Map metadata,
                 String unit,
                 BigDecimal level)
```

Constructs new `LevelData` instance with the specified arguments.

Parameters:

`timestamp` - The data timestamp.
`metadata` - The data metadata.
`unit` - The data unit.
`level` - The level value.

Method Detail

getLevel

```
public BigDecimal getLevel()
```

Returns `LevelData` value. The value type is `BigDecimal` instead of `double` to guarantee value accuracy.

Returns:

The `LevelData` value.

---

getUnit

```
public String getUnit()
```

Returns `LevelData` unit as it's specified in `org.osgi.service.dal.PropertyMetadata.UNITS` or `null` if the unit is missing.

Returns:

The value unit or `null` if the unit is missing.

---

equals

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

Two `LevelData` instances are equal if they contain equal metadata, timestamp, unit and level.

Overrides:

`equals` in class `org.osgi.service.dal.FunctionData`

Parameters:

`other` - The object to compare this data.

Returns:

`true` if this object is equivalent to the specified one.

See Also:

`org.osgi.service.dal.FunctionData.equals(java.lang.Object)`

---

hashCode

```
public int hashCode()
```

Returns the hash code for this `LevelData` object. The hash code is a sum of `org.osgi.service.dal.FunctionData.hashCode()`, `String.hashCode()` and `BigDecimal.hashCode()`, where `String.hashCode()` represents the unit hash code and `BigDecimal.hashCode()` represents the level hash code.

Overrides:

`hashCode` in class `org.osgi.service.dal.FunctionData`

Returns:

The hash code of this `LevelData` object.

See Also:

`org.osgi.service.dal.FunctionData.hashCode()`

---

compareTo

```
public int compareTo(Object o)
```

Compares this `LevelData` instance with the given argument. The argument can be:

- `BigDecimal` - the method returns the result of `BigDecimal.compareTo(Object)` for this instance level and the specified argument.
- `LevelData` - the method returns `-1` if metadata, timestamp or unit are not equivalent. Otherwise, the level is compared with the same rules as `BigDecimal` argument.
- `Map` - the map must be built according the rules of `LevelData(Map)`. Metadata, timestamp, unit and level are compared according `BigDecimal` and `LevelData` argument rules.

Specified by:

`compareTo` in interface `Comparable`

Parameters:

`o` - An argument to be compared.

Returns:

-1, 0 or 1 depending on the comparison rules.

Throws:

`ClassCastException` - If the method is called with `Map` and the field value types are not expected.
`IllegalArgumentException` - If the method is called with `Map` and the level is missing.
`NullPointerException` - If the argument is `null`.

See Also:

`Comparable.compareTo(java.lang.Object)`

---

8 Considered Alternatives

Currently, there are no alternatives.

9 Security Considerations

Currenty, the security is covered by OSGi Device Abstraction Layer.

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

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 End of Document