RFC 84 Monitoring

Confidential, Draft

35 Pages

Abstract

This document specifies a technical solution to perform monitoring of bundles running on the OSGi. Only local access (via Java API) is defined here, remote (via Management Tree operations) access of monitoring data is defined in RFC-87.

0 Document Information

0.1 Table of Contents

All Page Within This Box

0.2 Terminology and Document Conventions

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

Terminology can be found in the MEG Glossary [7].

0.3 Revision History

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

Revision	Date	Comments
Initial	05 28 2004	Initial version. Balázs Gődény, Nokia, balazs.godeny@nokia.com
V0.1	06 04 2004	Comments of the 06/02 call addressed. Comments from Peter Kriens addressed. Open issues section added. Balázs Gődény
V0.2	08 17 2004	API modified to better match the OMA trap proposal Deleted the DMT mapping chapter Balázs Gődény
V0.3	09 01 2004	Comments from the Boston meeting addressed Balázs Gődény
V0.4	10 05 2004	MonitorEventData class removed, event specification section added. Minor corrections. Balázs Gődény

All Page Within This Box

Revision	Date	Comments
V0.5	11 02 2004	Instant notification feature introduced: UpdateListener interface added Balázs Gődény
V0.6	11 12 2004	Monitorable.resetKpi() added. Clarification text added to several chapters. Sequence charts added. Balázs Gődény
V0.7	12.17 2004	KpiPermission added. Behaviour of KpiPermission checking explained in the JavaDocs. Small corrections. Balázs Gődény
V0.8	02.02.2005	Changes after first version of spec is done. Major changes: - The name KPI is changed to status variable - Object type is no longer allowed for KPIs - Monitorable instances are not accessible to the local admins - Added 'discover' and 'switchevent' actions to MonitorPermission - UpdateListener is renamed to MonitorListener Balázs Gődény
V0.9	03.16.2005	Comments from the Munnich discussion addressed. StatusVariable type int changed to long, float changed to double, boolean added. Balázs Gődény

1 Introduction

This document is part of the overall effort of the *Mobile Expert Group* (MEG) in the OSGi to come up with a technical solution for operational management of mobile device platforms based on CDC configuration of J2ME and OSGi. The effort is subdivided into several *Work Streams* (WS) as follows:

- Application model
- Device Management
- Policies
- Deployment

All Page Within This Box

It also encompasses two horizontal efforts, which manifest themselves in all vertical aspects of the platform:

- Security
- Non-functional requirements and features

This RFC document is devoted to consideration of one sub area of the Device Management: monitoring. It addresses the question of how an application or service running in the mobile device can make information about its internal state available for management entities, both local and remote.

2 Application Domain

See the Device Management RFC ([11]) for the description of the application domain.

3 Problem Description

Applications and services may publish status indicator variables (a.k.a. Key Performance Indicators, KPI) that management systems can query obtaining information about the behaviour of the application. There should be a standardized interface available for software components for publishing a list of available status variables and making status variable values available. There should be an interface where periodic update of these indicators can be initiated. Also there should be a method for remote management systems to issue these operations.

4 Requirements

The relevant use cases and requirements from the Device Management RFP [5] are copied here. For historical reasons status variables are called KPIs in this chapter.

4.1 Use cases

4.1.1 Retrieve application and service KPIs

4.1.1.1 Description

The Administrator or the User wants to inspect the behaviour of the application or service using the KPIs that the application or service published. The device management system is requested to return the values of a certain KPI or group of KPIs. The device management system returns the value(s) of the requested KPI(s)

4.1.1.2 Exceptions

- The application or service whose KPIs are requested is not installed
- The User or Administrator doesn't have right to access the KPIs of the application or service
- The KPI requested was not published by the application

4.1.2 Periodical update of the performance indicators

4.1.2.1 Description

The Administrator or User wants to inspect some KPI regularly to follow the behaviour of the application or service continuously. The device management system is requested to send the KPI value regularly to the caller. The device management system starts sending the KPI values at regular interval.

4.1.2.2 Exceptions

- The application or service whose KPIs are requested is not installed
- The User or Administrator doesn't have right to access the KPIs of the application or service
- The KPI requested was not published by the application

Policy doesn't allow to the requested regularity (requested update was too frequent, requested data too large, etc.)

4.2 Requirements

REQ-DM-02-03. There MAY be an interface between the application and the management system where the application can publish Key Performance Indicators (KPIs).

REQ-DM-02-05. It MAY be possible to retrieve KPIs of an application on the management server operator's request. For example if the user complains that the application doesn't work correctly, helpdesk personnel can examine the KPIs and decide an action.

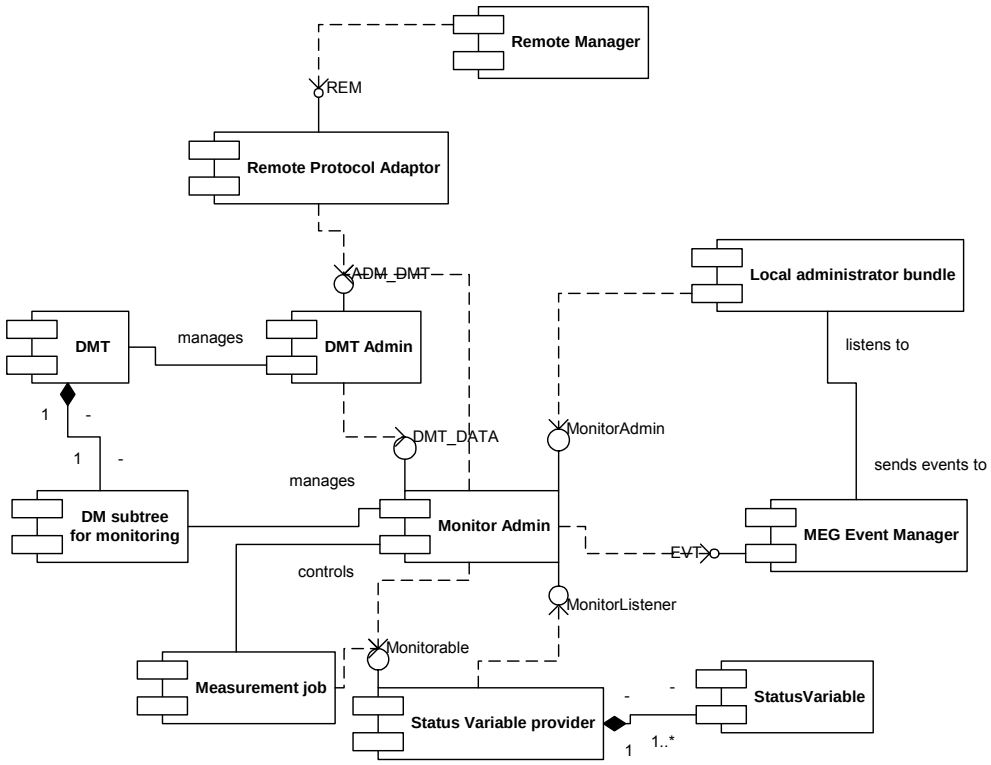
REQ-DM-03-05. It MAY be possible to launch a periodic diagnosis update operation. For example it is possible to request that certain KPI(s) be monitored for a certain time and the values be written into the log. The management server then collects the data when the operation is ready and the administrator personnel makes an analysis

All Page Within This Box

5 Technical Solution

5.1 Entities

The following figure shows the entities that play a role in the monitoring solution.



A short description of the entities follows. The modules and interfaces which are specified in this document are highlighted in bold font. Detailed description of other entities can be found in [4].

Status variable	Application-specific monitoring variable that a status variable provider (a monitorable entity) publishes through the management system in order to allow its status to be examined. Stores a state variable in one of the following formats: long, double, boolean or string. Contains information on the collection method of the variable, and the time when the value was measured.
Status variable provider	An application or service which makes status variables available for query through the **Monitorable** interface. Bundle programmers should implement this interface if they want to allow their bundles to be monitorable. Instances of this interface should register themselves at the OSGi Service Registry (not shown on picture). Optionally

All Page Within This Box

	it can provide instant notification to the Monitor Admin when the status of its status variables change.
Monitor Admin	A service which handles status variable query requests and measurement job control requests. It provides the **MonitorAdmin** interface as a convenient means for local managers to issue these requests, and also implements the DMT_DATA interface, where the DMTAdmin can issue the same requests in the form of DMT operations. For the latter it has to manage the monitoring related subtree of the DMT. It also provides the **MonotorListener** interface where Monitorable services can notify the Monitor Admin about changes in their status variables' state. Note that the Monitorable service instances are not accessible through this API.
Measurement job	An event based or time based periodic query of a given set of status variables. When a status variable value is updated within a measurement job the Monitor generates a **Monitoring Event** which can be received by local adminstrator bundles, or it sends a notification to remote management servers.
DM subtree for monitoring	A subtree of the Device Management Tree. A well defined data structure which represent all the status variables available in the system, and which also represent those running measurement jobs which were initiated remotely. By manipulating nodes the measurement jobs can be controlled remotely. The exact tree structure is defined in [8].
Local administrator bundle	A local management application which uses the MonitorAdmin interface to query status variables and to initiate measurement jobs. The application can receive Monitoring Events if it is registered at the Event Manager as a listener.
Remote manager, Remote Protocol Adaptor, DMTAdmin, DMT	A management server not residing on the OSGi service platform can query status variables and control measurement jobs through DMT manipulation commands. The commands reach the service platform through the REM interface where the Remote Protocol Adaptor forwards the calls to the DMTAdmin through the ADM_DMT interface, which eventually ends up at the Monitor through the DMT_DATA interface. The flow of events can also go the other way round. When a status variable is updated the Monitor Admin sends an instant notification to the DMT Admin (using it's AlertSender interface) which will send the notification (alert in OMA DM terminology) to the remote manager (not shown on the picture).

5.2 Status variable collection methods

3.3 b) in [10] lists the possible collection methods that a status variable might support. The collection method is independent of how (if and when) the reporting of the variables should happen. The collection method is part of the status variable's definition and can not be changed, whereas the reporting rules are defined by the entities interested in the value of the status variable. The same status variable can participate in more then one monitoring job, thus it might have more than one reporting rules defined.

The following sections explain the different collection methods available in the Monitor Admin interface.

All Page Within This Box

5.2.1 Cumulative counters (CC)

A numeric counter whose value can only increase, with the exception of the case when the status variable's reset() method is explicitely called. An example of a CC is a variable which stores the number of incoming SMSs handled by the protocol driver since it was started.

5.2.2 Gauge (GAUGE)

A numeric counter whose value can vary up or down. An example of a GAUGE is a variable which stores the current battery level percentage. Note that the current value of the counter is the current value of the status variable and not the difference between the current and previous values.

5.2.3 Discrete event registration (DER)

A not necessarily numeric status variable which can change when a certain event happens in the system. The event which fires the change of the status variable is typically not a monitoring event, but some other event like the arrival of an SMS. The definition of a DER counter contains an integer N which means how many events it takes for the counter to change its value. The most usual value for N is 1, but if N is greater than 1 then it means that the variable changes after each Nth event.

5.2.4 Status inspection (SI)

The most general, not necessarily numeric status variable. An example of an SI is a string variable which contains the name of the currently logged in user.

5.3 Monitoring jobs

A local administrator can issue a monitoring job, which means that it expresses its interest in the values of certain status variables. The initiator of the job can specify the rules which define when the measurement data will be reported back to him. There are two flavours of scheduling rules:

- The initiator can specify a frequency with which measurements must be taken and data reported back. For example the initiator can specify that the current value of status variable "`myapp/number_of_threads`" must be reported once every minute. The Monitor Admin in this case will query the status variable with the given frequency and the initiator of the job will receive monitoring events with the frequency he set up when the job was started.

- The initiator can request for instant notification whenever a value of the status variable change. In this case the administrator must make sure that the status variables he is interested in support this instant notification feature. The initiator of the job will receive monitoring events when the status variable changed. It is also possible to specify that monitoring events are sent only after every Nth change of the status variables value.

Monitoring jobs can be started also remotely by a management server through Device Management Tree operations. Because of limitations in the DMT structure, in the remote case the job can contain only one status variable. The monitoring job has a method which tells whether it was started locally or remotely.

5.4 Monitoring event

The Monitor Admin sends an event to the Event Manager whenever a status variable is updated. This can happen either because a Monitorable reported the change on the MonitorListener interface or when the status variable was explicitely reset to its starting value, or when the status variable is queried from within the job by the Monitor Admin,. Event sending in the first two cases can be switched on and off, but in the case of monitoring jobs it can not. Monitoring events are sent asynchronously.

All Page Within This Box

The event complies to the format specified in [12]. The topic of the event is “org.osgi.service.monitor”.

The properties of the event are:

Property	Type	Comment
mon.monitorable.pid	String	The unique identifier of the Monitorable service which updated a status variable.
mon.statusvariable.name	String	The name of the updated status variable.
mon.listener.id	String or String[ ]	Name or names representing the initiators of the monitoring job in which the status variable was updated. Listeners can use this field for filtering, so that they receive only events related to jobs initiated by them. If the event is fired because of a notification on the UpdateListener interface of the Monitor Admin (and not because of an update within a monitoring job) then this property is not present.
mon.statusvariable.value	String	The value of the status variable in string format. If the variable is of type long then this property must be generated as `Long.toString(longvalue)`. If it is of type double then this property must be generated as `Double.toString(floatvalue)`. If it is of type boolean then this property must be generated as `Boolean.toString(booleanvalue)`.

Note that the event which causes a status variable of DER type to be updated is not related to the monitoring events.

5.5 Operation

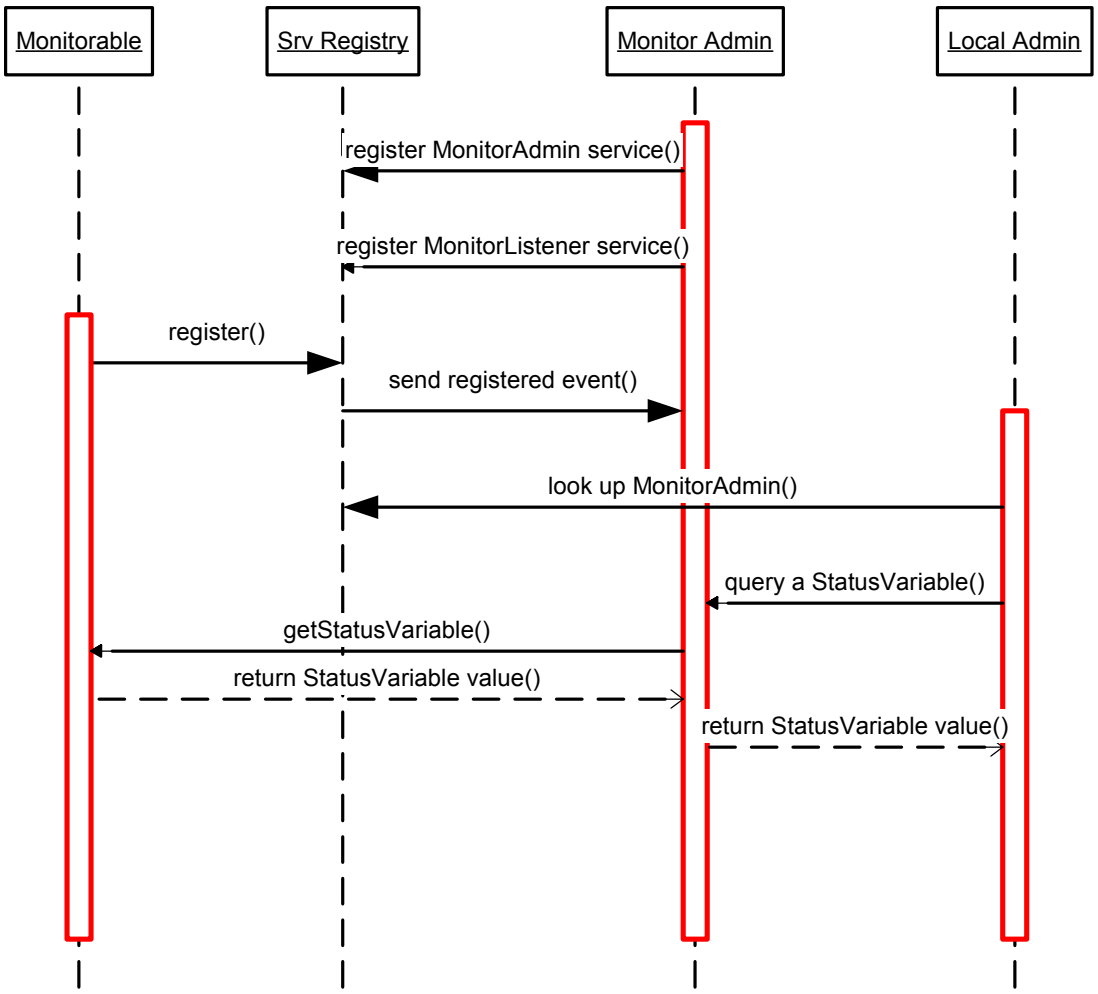
The following sequence charts show the flow of events in the most common use cases involving a local manager. Remote management is not discussed here, see [8].

Registering Monitorables, querying status variables and starting measurement jobs requires MonitorPermission. If the entity issuing the operation does not have this permission, a SecurityException is thrown. For simplicity, the charts do not show permission checks and the alternative sequence flows in cases where the required permission is not present.

5.5.1 Simple status variable request

In this scenario the local administrator bundle reads a value of a status variable published by a Monitorable service.

All Page Within This Box



1. During system startup the Monitor Admin component registers itself as a MonitorAdmin service and also as an MonitorListener service at the OSGi service registry. (The MonitorListener service will not be used in this scenario.)

2. The to-be-monitored bundle creates a Monitorable service which also registeres itself at the registry.

3. The framework notifies the Monitor Admin component of the registration of the Monitorable service.

4. The local administrator looks up the Monitor Admin from the registry.

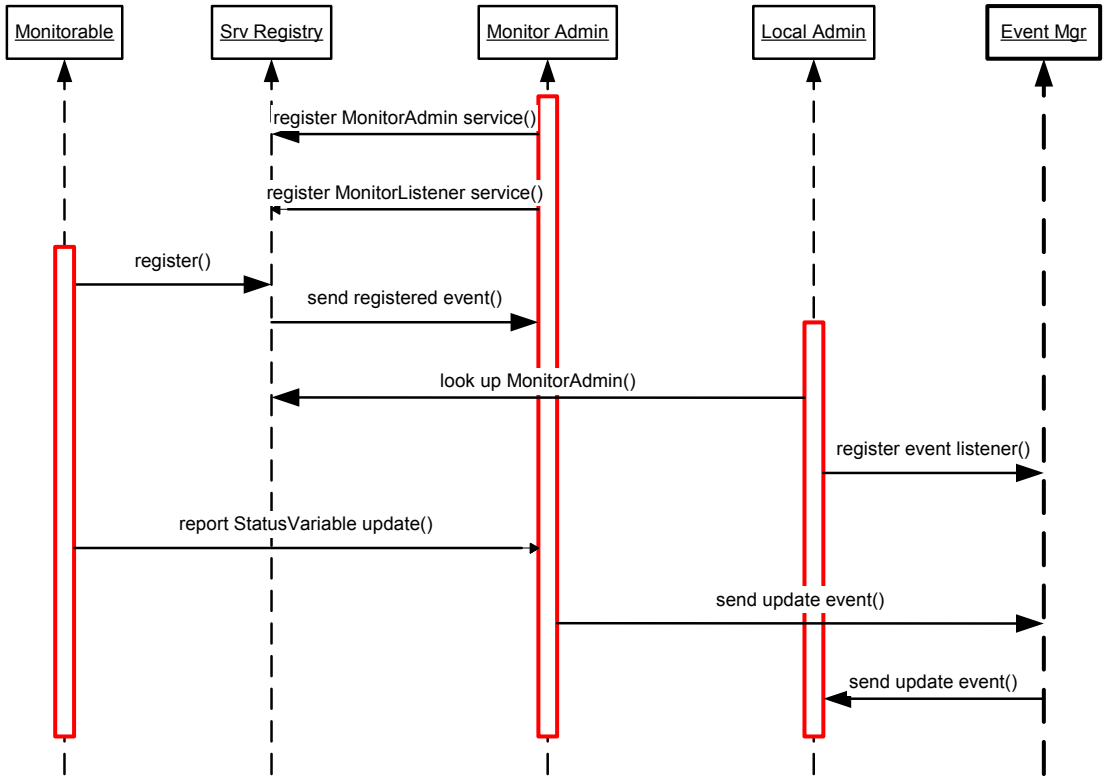
5. The local administrator queries a status variable value using the status variables unique identifier. It is assumed here that the administrator knows the ID of the status variable. Alternatively the administrator could query the list of registered Monitorable services from the registry and issue the getstatus variable call on a Monitorable. This alternative flow is not shown on the picture.

6. The Monitor Admin forwards the getKPI call to the Monitorable.

All Page Within This Box

7. The current value of the status variable is returned to the Monitor Admin and then to the local administrator.

5.5.2 Notification on update



In this scenario the local administrator receives events when a Monitorable updates its status variable. The first 4 steps are the same as in the previous scenario, their description is not repeated.

1. The local administrator registers itself as an event listener. It optionally can set up a filter defining which status variables it is interested in.

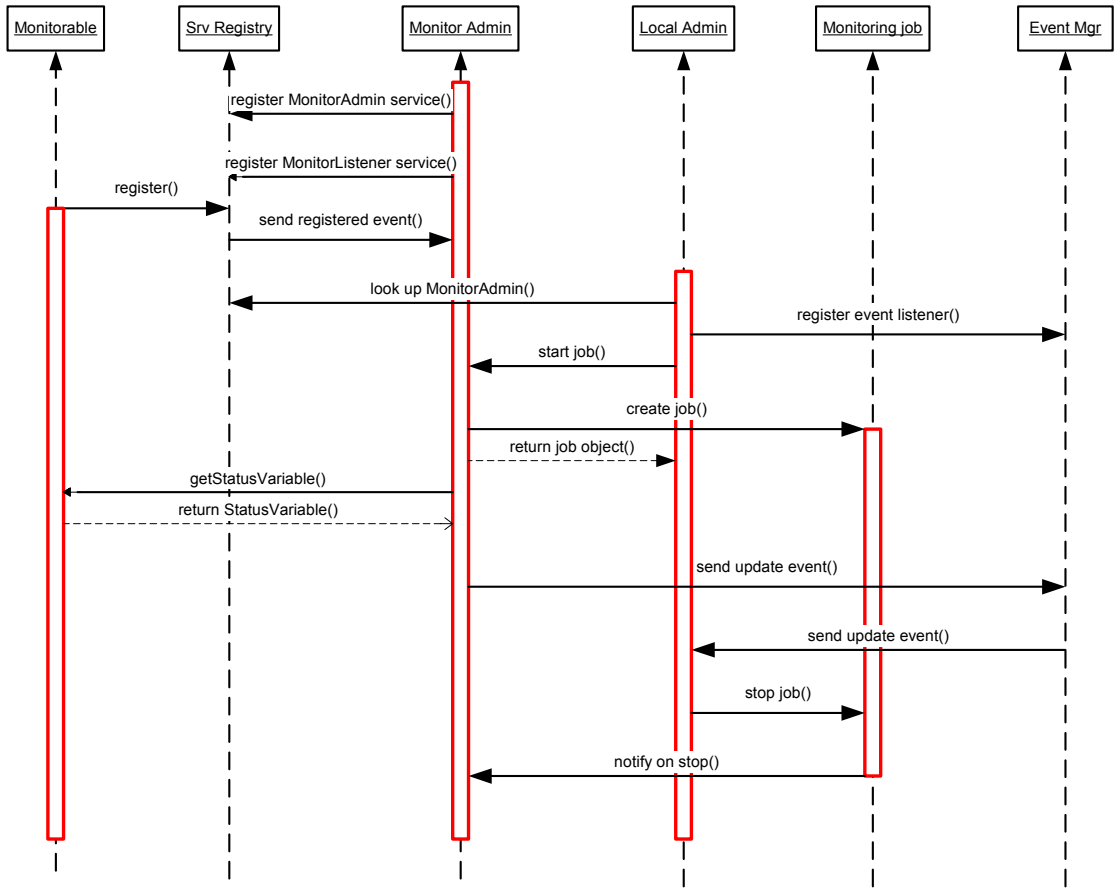
2. One of the status variables published by the Monitorable service is updated, the Monitorable reports the update on the UpdateListener service of the Monitor Admin component.

3. The Monitor Admin sends an event the event manager.

4. If the event matches the filter set up by the administrator (or no filter was used) it is forwarded to him. After receiving the event the administrator application may query the updated value of the status variable from the MonitorAdmin.

5.5.3 Time based monitoring job

All Page Within This Box



In this scenario the local administrator sets up a time based monitoring job and receives status variable updates periodically. The first 4 steps are the same as in the previous scenario, their description is not repeated.

1. The local administrator registers itself as an event listener. It can set up a filter containing a listener ID which will be used later to delivering the events to it.

2. The local administrator starts a monitoring job using the Monitor Admin interface. It specifies the status variables it is interested in, the frequency of the notifications it requires and optionally the number of samples it wants to receive.

3. The Monitor Admin creates the Monitoring Job object and returns it to the administrator.

4. The Monitor Admin queries the given status variables with the given frequency.

5. The Monitor Admin sends Monitoring Events The events contain the listener ID of the local administrator and so that it can receive them using the correct filter. After receiving the events the administrator application may query the updated value of the status variable from the MonitorAdmin.

6. Steps 4 and 5 are repeated until either the administrator explicitely stops the job or the Monitor Admin already sent the number of notifications the administrator was interested in.

All Page Within This Box

7. If the job was stopped explicitely then the Monitor Admin is notified so that it can delete the job from its list of active jobs.

5.5.4 Event based monitoring job

This scenario is the same as the previous one with these exceptions.

- In step 2 it is not possible to specify the number of reports the administrator wants to receive, so the job does not stop automatically: it has to be stopped explicitely. Instead, the administrator can specify that it is not interested in every update of the status variables, but only in the every Nth update.
- In step 4 the Monitor Admin does not query the status variable, instead it listens for updates from the Monitorables on its UpdateListener interface.
- In step 5 the Monitor Admin sends events only if it received the Nth (2*N, 3*N, etc) update of the status variable, where N was specified when the job was started. If the job contains more than one status variable then counting of Ns happens per status variable.

6 Java API

All classes and interfaces are in the `org.osgi.service.monitor` package

6.1 StatusVariable class

public final class **StatusVariable**
extends java.lang.Object

A StatusVariable object represents the value of a status variable taken with a certain collection method at a certain point of time. The type of the StatusVariable can be long, double, boolean or string.

Field Summary

static int	**CM_CC** Collection method type identifying 'Cumulative Counter' data collection.
static int	**CM_DER** Collection method type identifying 'Discrete Event Registration' data collection.
static int	**CM_GAUGE** Collection method type identifying 'Gauge' data collection.
static int	**CM_SI** Collection method type identifying 'Status Inspection' data collection.
static int	**TYPE_BOOLEAN** StatusVariable type identifying string data.
static int	**TYPE_DOUBLE** StatusVariable type identifying double data.

All Page Within This Box

static int	**TYPE_LONG** StatusVariable type identifying long data.
static int	**TYPE_STRING** StatusVariable type identifying string data.

Constructor Summary

StatusVariable(java.lang.String monitorableId, java.lang.String id, int cm, boolean data) Constructor for a StatusVariable of boolean type.	
StatusVariable(java.lang.String monitorableId, java.lang.String id, int cm, double data) Constructor for a StatusVariable of double type.	
StatusVariable(java.lang.String monitorableId, java.lang.String id, int cm, long data) Constructor for a StatusVariable of long type.	
StatusVariable(java.lang.String monitorableId, java.lang.String id, int cm, java.lang.String data) Constructor for a StatusVariable of String type.	

Method Summary

boolean	**equals**(java.lang.Object obj) Compares the specified object with this StatusVariable.
boolean	**getBoolean**() Returns the StatusVariable value if its type is boolean.
int	**getCollectionMethod**() Returns the collection method of this StatusVariable.
double	**getDouble**() Returns the StatusVariable value if its type is double.
java.lang.String	**getID**() Returns the name of this StatusVariable.
long	**getLong**() Returns the StatusVariable value if its type is long.
java.lang.String	**getString**() Returns the StatusVariable value if its type is String.
java.util.Date	**getTimeStamp**() Returns the time when the StatusVariable value was queried.
int	**getType**() Returns information on the data type of this StatusVariable.
int	**hashCode**() Returns the hash code value for this StatusVariable.
java.lang.String	**toString**() Returns a string representation of this StatusVariable.

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

All Page Within This Box

Field Detail

6.1.1 TYPE_LONG

public static final int **TYPE_LONG**

StatusVariable type identifying long data.

See Also:

Constant Field Values

6.1.2 TYPE_DOUBLE

public static final int **TYPE_DOUBLE**

StatusVariable type identifying double data.

See Also:

Constant Field Values

6.1.3 TYPE_STRING

public static final int **TYPE_STRING**

StatusVariable type identifying string data.

See Also:

Constant Field Values

6.1.4 TYPE_BOOLEAN

public static final int **TYPE_BOOLEAN**

StatusVariable type identifying string data.

See Also:

Constant Field Values

6.1.5 CM_CC

public static final int **CM_CC**

Collection method type identifying 'Cumulative Counter' data collection.

See Also:

Constant Field Values

6.1.6 CM_DER

public static final int **CM_DER**

Collection method type identifying 'Discrete Event Registration' data collection.

See Also:

Constant Field Values

6.1.7 CM_GAUGE

public static final int **CM_GAUGE**

Collection method type identifying 'Gauge' data collection.

See Also:

Constant Field Values

6.1.8 CM_SI

public static final int **CM_SI**

Collection method type identifying 'Status Inspection' data collection.

See Also:
Constant Field Values

Constructor Detail

6.1.9 StatusVariable

public **StatusVariable**(java.lang.String monitorableId,
java.lang.String id,
int cm,
long data)

Constructor for a StatusVariable of long type.

Parameters:

monitorableId - The identifier of the monitorable service that this StatusVariable belongs to
id - The identifier of the StatusVariable
cm - Collection method, should be one of the CM_ constants
data - The long value of the StatusVariable

Throws:

java.lang.IllegalArgumentException - if any of the ID parameters contains invalid characters or if cm is not one of the collection method constants
java.lang.NullPointerException - if any of the ID parameters is null

---

6.1.10 StatusVariable

public **StatusVariable**(java.lang.String monitorableId,
java.lang.String id,
int cm,
double data)

Constructor for a StatusVariable of double type.

Parameters:

monitorableId - The identifier of the monitorable service that this StatusVariable belongs to
id - The identifier of the StatusVariable
cm - Collection method, should be one of the CM_ constants
data - The double value of the StatusVariable

Throws:

java.lang.IllegalArgumentException - if any of the ID parameters contains invalid characters or if cm is not one of the collection method constants
java.lang.NullPointerException - if any of the ID parameters is null

---

6.1.11 StatusVariable

public **StatusVariable**(java.lang.String monitorableId,
java.lang.String id,
int cm,
boolean data)

Constructor for a StatusVariable of boolean type.

Parameters:

monitorableId - The identifier of the monitorable service that this StatusVariable belongs to
id - The identifier of the StatusVariable
cm - Collection method, should be one of the CM_ constants
data - The boolean value of the StatusVariable

Throws:

java.lang.IllegalArgumentException - if any of the ID parameters contains invalid characters or if cm is not one of the collection method constants
java.lang.NullPointerException - if any of the ID parameters is null

All Page Within This Box

6.1.12 StatusVariable

public **StatusVariable**(java.lang.String monitorableId,
java.lang.String id,
int cm,
java.lang.String data)

Constructor for a StatusVariable of String type.

Parameters:

monitorableId - The identifier of the monitorable service that this StatusVariable belongs to
id - The identifier of the StatusVariable
cm - Collection method, should be one of the CM_ constants
data - The string value of the StatusVariable

Throws:

java.lang.IllegalArgumentException - if any of the ID parameters contains invalid characters or if cm is not one of the collection method constants
java.lang.NullPointerException - if any of the ID parameters is null

Method Detail

6.1.13 getID

public java.lang.String **getID**()

Returns the name of this StatusVariable. A StatusVariable name is unique within the scope of a Monitorable. A StatusVariable name must not contain the Reserved characters described in 2.2 of RFC-2396 (URI Generic Syntax).

Returns:

the name of this StatusVariable

6.1.14 getType

public int **getType**()

Returns information on the data type of this StatusVariable.

Returns:

one value of the set of type constants

6.1.15 getTimeStamp

public java.util.Date **getTimeStamp**()

Returns the time when the StatusVariable value was queried. The StatusVariable's value is set when the getStatusVariable() or getStatusVariables() methods are called on the Monitorable object.

Returns:

the time when the StatusVariable value was queried

6.1.16 getString

public java.lang.String **getString**()
throws java.lang.IllegalStateException

Returns the StatusVariable value if its type is String.

Returns:

the StatusVariable value as a string

Throws:

java.lang.IllegalStateException - if the type of the StatusVariable is not String

---

All Page Within This Box

6.1.17 getLong

public long **getLong**()
throws java.lang.IllegalStateException

Returns the StatusVariable value if its type is long.

Returns:

the StatusVariable value as an long

Throws:

java.lang.IllegalStateException - if the type of this StatusVariable is not long

---

6.1.18 getDouble

public double **getDouble**()
throws java.lang.IllegalStateException

Returns the StatusVariable value if its type is double.

Returns:

the StatusVariable value as a double

Throws:

java.lang.IllegalStateException - if the type of this StatusVariable is not double

---

6.1.19 getBoolean

public boolean **getBoolean**()
throws java.lang.IllegalStateException

Returns the StatusVariable value if its type is boolean.

Returns:

the StatusVariable value as a boolean

Throws:

java.lang.IllegalStateException - if the type of this StatusVariable is not double

---

6.1.20 getCollectionMethod

public int **getCollectionMethod**()

Returns the collection method of this StatusVariable. See section 3.3 b) in [ETSI TS 132 403]

Returns:

one of the CM_ constants

6.1.21 equals

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

Compares the specified object with this StatusVariable. Two StatusVariable objects are considered equal if their full path, collection method and type are identical, and the data (selected by their type) is equal.

Parameters:

obj - the object to compare with this StatusVariable

Returns:

true if the argument represents the same StatusVariable as this object

---

6.1.22 hashCode

public int **hashCode**()

Returns the hash code value for this StatusVariable. The hash code is calculated based on the full path, collection method and value of the StatusVariable.

6.1.23 toString

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

All Page Within This Box

Returns a string representation of this StatusVariable. The returned string contains the full path, the collection method, the exact time of creation, the type and the value of the StatusVariable in the following format: StatusVariable(<path>, <cm>, <timestamp>, <type>, <value>)

Returns:

the string representation of this StatusVariable

6.2 Monitorable interface

public interface **Monitorable**

A Monitorable can provide information about itself in the form of StatusVariables. Instances of this interface should register themselves at the OSGi Service Registry. The MonitorAdmin listens to the registration of Monitorable services, and makes the information they provide available also through the Device Management Tree (DMT).

The monitorable service is identified by its PID string which must not contain the Reserved characters described in 2.2 of RFC-2396 (URI Generic Syntax). Also the length of the PID should be kept as small as possible. The PID will be used as a node name in the DMT and certain DMT implementations may have limits on node name length. The length limit is not specified in any standard, it is recommended not to use names longer than 20 characters.

A Monitorable may optionally support sending notifications when the status of its StatusVariables change.

Publishing StatusVariables requires the presence of the MonitorPermission with the publish action string. This permission, however, is not checked during registration of the Monitorable service. Instead, the MonitorAdmin implemenatation must make sure that when a StatusVariable is queried, it is shown only if the Monitorable is authorized to publish the given StatusVariable.

Method Summary

java.lang.String	**getDescription**(java.lang.String id) Returns a human readable description of a StatusVariable.
StatusVariable	**getStatusVariable**(java.lang.String id) Returns the StatusVariable object addressed by its identifier.
java.lang.String[]	**getStatusVariableNames**() Returns the list of StatusVariable identifiers published by this Monitorable.
boolean	**notifiesOnChange**(java.lang.String id) Tells whether the StatusVariable provider is able to send instant notifications when the given StatusVariable changes.
boolean	**resetStatusVariable**(java.lang.String id) Issues a request to reset a given StatusVariable.

Method Detail

6.2.1 getStatusVariableNames

public java.lang.String[] **getStatusVariableNames**()

Returns the list of StatusVariable identifiers published by this Monitorable. A StatusVariable name is unique within the scope of a Monitorable. The array contains the elements in no particular order.

Returns:

All Page Within This Box

the name of StatusVariables published by this object

6.2.2 getStatusVariable

public StatusVariable **getStatusVariable**(java.lang.String id)
throws java.lang.IllegalArgumentException

Returns the StatusVariable object addressed by its identifier. The StatusVariable will hold the value taken at the time of this method call.

Parameters:

id - the identifier of the StatusVariable. The identifier does not contain the Monitorable_id, i.e. this is the name and not the path of the status variable.

Returns:

the StatusVariable object

Throws:

java.lang.IllegalArgumentException - if the name points to a non existing StatusVariable

---

6.2.3 notifiesOnChange

public boolean **notifiesOnChange**(java.lang.String id)
throws java.lang.IllegalArgumentException

Tells whether the StatusVariable provider is able to send instant notifications when the given StatusVariable changes. If the Monitorable supports sending change updates it must notify the MonitorListener when the value of the StatusVariable changes. The Monitorable finds the MonitorListener service through the Service Registry.

Parameters:

id - the identifier of the StatusVariable. The identifier does not contain the Monitorable_id, i.e. this is the name and not the path of the status variable.

Returns:

true if the Monitorable can send notification when the given StatusVariable chages, false otherwise

Throws:

java.lang.IllegalArgumentException - if the path is invalid or points to a non existing StatusVariable

---

6.2.4 resetStatusVariable

public boolean **resetStatusVariable**(java.lang.String id)
throws java.lang.IllegalArgumentException

Issues a request to reset a given StatusVariable. Depending on the semantics of the StatusVariable this call may or may not succeed: it makes sense to reset a counter to its starting value, but e.g. a StatusVariable of type String might not have a meaningful default value. Note that for numeric StatusVariables the starting value may not necessarily be 0. Resetting a StatusVariable triggers a monitor event.

Parameters:

id - the identifier of the StatusVariable.

Returns:

true if the Monitorable could successfully reset the given StatusVariable, false otherwise

Throws:

java.lang.IllegalArgumentException - if the id points to a non existing StatusVariable

---

6.2.5 getDescription

public java.lang.String **getDescription**(java.lang.String id)

Returns a human readable description of a StatusVariable. This can be used by management systems on their GUI. Null return value is allowed.

Parameters:

id - Identifier of a StatusVariable published by this Monitorable

All Page Within This Box

Returns:
the human readable description of this StatusVariable or null if it is not set
Throws:
java.lang.IllegalArgumentException - if the path points to a non existing StatusVariable

6.3 MonitorAdmin interface

public interface **MonitorAdmin**

A MonitorAdmin implementation handles StatusVariable query requests and measurement job control requests.

Note that an alternative but not recommended way of obtaining StatusVariables is that applications having the required ServicePermissions can query the list of Monitorable services from the service registry and then query the list of StatusVariable names from the Monitorable services. This way all services which publish StatusVariables will be returned regardless of whether they do or do not hold the necessary MonitorPermission for publishing StatusVariables. By using the MonitorAdmin to obtain the StatusVariables it is guaranteed that only those Monitorable services will be accessed who are authorized to publish StatusVariables. It is the responsibility of the MonitorAdmin implementation to check the required permissions and show only those services which pass this check.

Method Summary

java.lang.String[]	**getMonitorableNames**() Returns the names of the Monitorable services that are currently registered.
MonitoringJob[]	**getRunningJobs**() Returns the list of currently running Monitoring Jobs.
StatusVariable	**getStatusVariable**(java.lang.String path) Returns a StatusVariable addressed by its ID in [Monitorable_ID]/[StatusVariable_ID] format.
java.lang.String[]	**getStatusVariableNames**(java.lang.String monitorableId) Returns the list of StatusVariable names published by a Monitorable instance.
StatusVariable[]	**getStatusVariables**(java.lang.String monitorableId) Returns all the StatusVariable objects published by a Monitorable instance.
boolean	**resetStatusVariable**(java.lang.String path) Issues a request to reset a given StatusVariable.
MonitoringJob	**startJob**(java.lang.String initiator, java.lang.String[] statusVariables, int count) Starts a change based Monitoring Job with the parameters provided.
MonitoringJob	**startScheduledJob**(java.lang.String initiator, java.lang.String[] statusVariables, int schedule, int count) Starts a time based Monitoring Job with the parameters provided.
void	**switchEvents**(java.lang.String path, boolean on) Switches event sending on or off for certain StatusVariable(s).

Method Detail

6.3.1 getStatusVariable

public StatusVariable **getStatusVariable**(java.lang.String path)

All Page Within This Box

throws java.lang.IllegalArgumentException

Returns a StatusVariable addressed by its ID in [Monitorable_ID]/[StatusVariable_ID] format. The entity which queries a StatusVariable needs to hold MonitorPermission for the given target with the read action present.

Parameters:

path - the full path of the StatusVariable in [Monitorable_ID]/[StatusVariable_ID] format

Returns:

the StatusVariable object

Throws:

java.lang.IllegalArgumentException - if the path is invalid or points to a non-existing StatusVariable

java.lang.SecurityException - if the caller does not hold a MonitorPermission for the StatusVariable specified by path with the read action present

---

6.3.2 getMonitorableNames

public java.lang.String[] **getMonitorableNames**()

Returns the names of the Monitorable services that are currently registered. The returned array contains the names in alphabetical order. For security reasons this method should be used instead of querying the monitorable services from the service registry. The Monitorable instances are not accessible through the MonitorAdmin.

Returns:

the array of Monitorable names or null if none are registered

Throws:

java.lang.SecurityException - if the caller does not hold a MonitorPermission with the discover action present. The target field must be "*/*".

6.3.3 getStatusVariables

public StatusVariable[] **getStatusVariables**(java.lang.String monitorableId)

Returns all the StatusVariable objects published by a Monitorable instance. The StatusVariables will hold the values taken at the time of this method call. The array contains the elements in no particular order.

The entity which queries the StatusVariable list needs to hold MonitorPermission with the read action present. The target field of the permission must match all the StatusVariables published by the Monitorable, e.g. it may be [Monitorable_ID]/*.

Parameters:

monitorableId - The identifier of a Monitorable instance

Returns:

the StatusVariable objects published by the specified Monitorable

Throws:

java.lang.SecurityException - if the caller does not hold MonitorPermission with the read action or if there is any StatusVariable published by the Monitorable which is not allowed to be read as per the target field of the permission

java.lang.IllegalArgumentException - if the monitorableId is invalid or points to a non-existing Monitorable

---

6.3.4 getStatusVariableNames

public java.lang.String[] **getStatusVariableNames**(java.lang.String monitorableId)

Returns the list of StatusVariable names published by a Monitorable instance. The array contains the elements in alphabetical order.

The entity which queries the StatusVariable list needs to hold MonitorPermission with the discover action present. The target field of the permission must match the identifier of the Monitorable service.

Parameters:

All Page Within This Box

monitorableId - The identifier of a Monitorable instance
Returns:
the names of the StatusVariable objects published by the specified Monitorable
Throws:
java.lang.SecurityException - if the caller does not hold MonitorPermission with the discover action or if the StatusVariables published by the given Monitorable are not allowed to be discovered as per the target field of the permission
java.lang.IllegalArgumentException - if the monitorableId is invalid or points to a non-existing Monitorable

---

6.3.5 switchEvents

public void **switchEvents**(java.lang.String path,
boolean on)

Switches event sending on or off for certain StatusVariable(s). When the MonitorAdmin is notified about a StatusVariable being updated it sends an event unless this feature is switched off. Note that events within a monitoring job can not be switched off. It is not required that the event sending state of the StatusVariables are persistently stored. When a StatusVariable is registered it's event sending state is ON by default.

Usage of the "*" wildcard is allowed in the path argument of this method as a convenience feature. The semantics of the wildcard is that it stands for any matching StatusVariable at the time of the method call, it does not effect the event sending status of StatusVariables which are not yet registered. As an example, when the switchEvents("MyMonitorable/*", false) method is executed, event sending from all StatusVariables of the MyMonitorable service are switched off. However, if the MyMonitorable service starts to publish a new StatusVariable later, it's event sending status is on by default.

Parameters:
path - The identifier of the StatusVariable(s) in [Monitorable_id]/[StatusVariable_id] format. The "*" wildcard is allowed in both path fragments.
on - If false then event sending is switched off, if true then it is switched on.
Throws:
java.lang.SecurityException - if the caller does not hold MonitorPermission with the switchevents action or if there is any StatusVariable in the path field for which it is not allowed to switch event sending on or off as per the target field of the permission
java.lang.IllegalArgumentException - if the path is invalid or points to a non-existing StatusVariable

---

6.3.6 resetStatusVariable

public boolean **resetStatusVariable**(java.lang.String path)
throws java.lang.IllegalArgumentException

Issues a request to reset a given StatusVariable. Depending on the semantics of the StatusVariable this call may or may not succeed: it makes sense to reset a counter to its starting value, but e.g. a StatusVariable of type String might not have a meaningful default value. Note that for numeric StatusVariables the starting value may not necessarily be 0. Resetting a StatusVariable triggers a monitor event if the StatusVariable supports update notifications.

The entity that wants to reset the StatusVariable needs to hold MonitorPermission with the reset action present. The target field of the permission must match the StatusVariable name to be reset.

Parameters:
path - the identifier of the StatusVariable in [Monitorable_id]/[StatusVariable_id] format.
Returns:
true if the Monitorable could successfully reset the given StatusVariable, false otherwise
Throws:
java.lang.IllegalArgumentException - if the path is invalid or points to a non existing StatusVariable

java.lang.SecurityException - if the caller does not hold MonitorPermission with the reset action or if the specified StatusVariable is not allowed to be reset as per the target field of the permission

---

6.3.7 startScheduledJob

public MonitoringJob **startScheduledJob**(java.lang.String initiator,
java.lang.String[] statusVariables,
int schedule,
int count)
throws java.lang.IllegalArgumentException

Starts a time based Monitoring Job with the parameters provided. Monitoring events will be sent according to the specified schedule. All specified StatusVariables must exist when the job is started. The initiator string is used in the mon.listener.id field of all events triggered by the job, to allow filtering the events based on the initiator.

The entity which initiates a Monitoring Job needs to hold MonitorPermission for all the specified target StatusVariables with the startjob action present. If the permission's action string specifies a minimal sampling interval then the schedule parameter should be at least as great as the value in the action string.

Parameters:

initiator - the identifier of the entity that initiated the job

statusVariables - List of StatusVariables to be monitored. The StatusVariable names must be given in [Monitorable_PID]/[StatusVariable_ID] format.

schedule - The time in seconds between two measurements. It must be greater than 0. The first measurement will be taken when the timer expires for the first time, not when this method is called.

count - The number of measurements to be taken, or 0 if the measurement must run infinitely.

Returns:

the successfully started job object

Throws:

java.lang.IllegalArgumentException - if the list of StatusVariable names contains a non-existing StatusVariable or the initiator, schedule or count parameters are invalid

java.lang.SecurityException - if the caller does not hold MonitorPermission for all the specified StatusVariables, with the startjob action present, or if the permission does not allow starting the job with the given frequency

---

6.3.8 startJob

public MonitoringJob **startJob**(java.lang.String initiator,
java.lang.String[] statusVariables,
int count)
throws java.lang.IllegalArgumentException

Starts a change based Monitoring Job with the parameters provided. Monitoring events will be sent when the StatusVariables of this job are updated. All specified StatusVariables must exist when the job is started, and all must support update notifications. The initiator string is used in the mon.listener.id field of all events triggered by the job, to allow filtering the events based on the initiator.

The entity which initiates a Monitoring Job needs to hold MonitorPermission for all the specified target StatusVariables with the startjob action present.

Parameters:

initiator - the identifier of the entity that initiated the job

statusVariables - List of StatusVariables to be monitored. The StatusVariable names must be given in [Monitorable_PID]/[StatusVariable_ID] format.

count - A positive integer specifying the number of changes that must happen to a StatusVariable before a new notification is sent.

Returns:

All Page Within This Box

the successfully started job object
Throws:
java.lang.IllegalArgumentException - if the list of StatusVariable names contains a non-existing StatusVariable or one that does not support notifications, or if the initiator or count parameters are invalid
java.lang.SecurityException - if the caller does not hold MonitorPermission for all the specified StatusVariables, with the startjob action present

---

6.3.9 getRunningJobs

public MonitoringJob[] **getRunningJobs**()
Returns the list of currently running Monitoring Jobs.
Returns:
the list of running jobs

6.4 MonitoringJob interface

public interface **MonitoringJob**

A Monitoring Job is a request for scheduled or event based notifications on update of a set of StatusVariables. The job is a data structure that holds a non-empty list of StatusVariable names, an identification of the initiator of the job, and the sampling parameters. There are two kinds of monitoring jobs: time based and change based. Time based jobs take samples of all StatusVariables with a specified frequency. The number of samples to be taken before the job finishes may be specified. Change based jobs are only interested in the changes of the monitored StatusVariables. In this case, the number of changes that must take place between two notifications can be specified.
The job can be started on the MonitorAdmin interface. Running the job (querying the StatusVariables, listening to changes, and sending out notifications on updates) is the task of the MonitorAdmin implementation.
Whether a monitoring job keeps track dynamically of the StatusVariables it monitors is not specified. This means that if we monitor a StatusVariable of a Monitorable service which disappears and later reappears then it is implementation specific whether we still receive updates of the StatusVariable changes or not.

Method Summary

java.lang.String	**getInitiator**() Returns the identitifier of the principal who initiated the job.
int	**getReportCount**() Returns the number of times MonitorAdmin will query the StatusVariables (for time based jobs), or the number of changes of a StatusVariable between notifications (for change based jobs).
long	**getSchedule**() Returns the delay (in seconds) between two samples.
java.lang.String[]	**getStatusVariableNames**() Returns the list of StatusVariable names that are the targets of this measurement job.
boolean	**isLocal**() Returns whether the job was started locally or remotely.
void	**stop**() Stops a Monitoring Job.

All Page Within This Box

Method Detail

6.4.1 stop

public void **stop**()

Stops a Monitoring Job. Note that a time based job can also stop automatically if the specified number of samples have been taken.

6.4.2 getInitiator

public java.lang.String **getInitiator**()

Returns the identitifier of the principal who initiated the job. This is set at the time when startJob() is called at the MonitorAdmin interface. This string holds the ServerID if the operation was initiated from a remote manager, or an arbitrary ID of the initiator entity in the local case (used for addressing notification events).

Returns:

the ID of the initiator

6.4.3 getStatusVariableNames

public java.lang.String[] **getStatusVariableNames**()

Returns the list of StatusVariable names that are the targets of this measurement job. For time based jobs, the MonitorAdmin will iterate through this list and query all StatusVariables when its timer set by the job's frequency rate expires.

Returns:

the target list of the measurement job in [Monitorable_ID]/[StatusVariable_ID] format

---

6.4.4 getSchedule

public long **getSchedule**()

Returns the delay (in seconds) between two samples. If this call returns N (greater than 0) then the MonitorAdmin queries each StatusVariable that belongs to this job every N seconds. The value 0 means that the job is not scheduled but event based: in this case instant notification on changes is requested (at every nth change of the value, as specified by the report count parameter).

Returns:

the delay (in seconds) between samples, or 0 for change based jobs

---

6.4.5 getReportCount

public int **getReportCount**()

Returns the number of times MonitorAdmin will query the StatusVariables (for time based jobs), or the number of changes of a StatusVariable between notifications (for change based jobs). Time based jobs with non-zero report count will take getReportCount()*getSchedule() time to finish. Time based jobs with 0 report count and change based jobs do not stop automatically, but all jobs can be stopped with the stop() method.

Returns:

the number of measurements to be taken, or the number of changes between notifications

---

6.4.6 isLocal

public boolean **isLocal**()

Returns whether the job was started locally or remotely.

Returns:

true if the job was started from the local device, false if the job was initiated from a remote management server through the device management tree

All Page Within This Box

6.5 MonitorListener interface

public interface **MonitorListener**

The MonitorListener is used by Monitorable services to send notifications when a StatusVariable value is changed. The MonitorListener should register itself as a service at the OSGi Service Registry. This interface is implemented by the Monitor Admin component.

Method Summary

void	**updated**(StatusVariable statusVariable) Callback for notification of a StatusVariable change.

Method Detail

6.5.1 updated

public void **updated**(StatusVariable statusVariable)

Callback for notification of a StatusVariable change.

Parameters:

statusVariable - the StatusVariable which has changed

6.6 MonitorPermission class

public class **MonitorPermission**

extends java.security.Permission

Indicates the callers authority to publish, discover, read or reset StatusVariables, to switch event sending on or off or to start monitoring jobs. The target of the permission is the identifier of the StatusVariable, the action can be read,publish, reset,startjob,switchevents, discover, or the combination of these separated by commas.

See Also:

Serialized Form

Field Summary

static java.lang.String	**DISCOVER** Holders of MonitorPermission with the discover action present are allowed to list the currently registered Monitorable services and the StatusVariables a Monitorable publishes.
static java.lang.String	**PUBLISH** Holders of MonitorPermission with the publish action present are Monitorable services that are allowed to publish the StatusVariables specified in the permission's target field.
static java.lang.String	**READ** Holders of MonitorPermission with the read action present are allowed to read the value of the StatusVariables specified in the permission's target field.
static java.lang.String	**RESET** Holders of MonitorPermission with the reset action present are allowed to reset the value of the StatusVariables specified in the permission's target field.
static java.lang.String	**STARTJOB** Holders of MonitorPermission with the startjob action present are allowed to initiate monitoring jobs involvingthe the StatusVariables specified in the permission's target field.
static java.lang.String	**SWITCHEVENTS**

All Page Within This Box

	Holders of MonitorPermission with the switchevents action present are allowed to switch event sending on or off for the value of the StatusVariables specified in the permission's target field.

Constructor Summary

MonitorPermission(java.lang.String statusVariable, java.lang.String actions) Create a MonitorPermission object, specifying the target and actions.	

Method Summary

boolean	**equals**(java.lang.Object o) Determines the equality of two MonitorPermission objects.
java.lang.String	**getActions**() Get the action string associated with this permission
int	**hashCode**() Create an integer hash of the object.
boolean	**implies**(java.security.Permission p) Determines if the specified permission is implied by this permission.

Methods inherited from class java.security.Permission
checkGuard, getName, newPermissionCollection, toString

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

Field Detail

6.6.1 READ

public static final java.lang.String **READ**

Holders of MonitorPermission with the read action present are allowed to read the value of the StatusVariables specified in the permission's target field.

See Also:

Constant Field Values

6.6.2 RESET

public static final java.lang.String **RESET**

Holders of MonitorPermission with the reset action present are allowed to reset the value of the StatusVariables specified in the permission's target field.

See Also:

Constant Field Values

6.6.3 PUBLISH

public static final java.lang.String **PUBLISH**

All Page Within This Box

Holders of MonitorPermission with the publish action present are Monitorable services that are allowed to publish the StatusVariables specified in the permission's target field.

See Also:

Constant Field Values

6.6.4 STARTJOB

public static final java.lang.String **STARTJOB**

Holders of MonitorPermission with the startjob action present are allowed to initiate monitoring jobs involvingthe the StatusVariables specified in the permission's target field.

A minimal sampling interval can be optionally defined in the following form: startjob:n. Here n is the allowed minimal value of the schedule parameter of time based monitoring jobs the holder of this permission is allowed to initiate. If n is not specified or 0 then the holder of this permission is allowed to start monitoring jobs specifying any frequency.

See Also:

Constant Field Values

6.6.5 DISCOVER

public static final java.lang.String **DISCOVER**

Holders of MonitorPermission with the discover action present are allowed to list the currently registered Monitorable services and the StatusVariables a Monitorable publishes. Discover rights are implied by read rights.

To list all Monitorables currently registered the permission's target field must be */*. To list the StatusVariables of a given Monitorable, the target field must be [M_PID]/* where M_PID matches the PID of the Monitorable service.

See Also:

Constant Field Values

6.6.6 SWITCHEVENTS

public static final java.lang.String **SWITCHEVENTS**

Holders of MonitorPermission with the switchevents action present are allowed to switch event sending on or off for the value of the StatusVariables specified in the permission's target field.

See Also:

Constant Field Values

Constructor Detail

6.6.7 MonitorPermission

public **MonitorPermission**(java.lang.String statusVariable,
java.lang.String actions)

Create a MonitorPermission object, specifying the target and actions.

Parameters:

statusVariable - The identifier of the StatusVariable in [Monitorable_id]/[StatusVariable_id] format. The semantics of this field is slightly different depending on the action field. See the description of the action fields.

The wildcard * is allowed in both fragments of the target string, but only at the end of the fragments. It is not allowed to use a wildcard in the first fragment but not to use it in the second fragment.

The following targets are valid: com.mycomp.myapp/queue_length, com.mycomp.*/queue*, com.mycomp.myapp/*,*/*.

All Page Within This Box

The following targets are invalid: *.myapp/queue_length, com.*.myapp/*,*, */queue_length.
actions - The allowed action(s): read, publish,startjob, reset,switchevents, discover or the combination of these separated by commas.

Method Detail

6.6.8 hashCode

public int **hashCode**()

Create an integer hash of the object. The hash codes of StatusVariablePermissions p1 and p2 are the same if p1.equals(p2)

Returns:

the hash of the object

6.6.9 equals

public boolean **equals**(java.lang.Object o)

Determines the equality of two MonitorPermission objects. Two MonitorPermission objects are equal if their target strings are equal and the same set of actions are listed in their action strings.

Parameters:

o - the object being compared for equality with this object

Returns:

true if the two permissions are equal

6.6.10 getActions

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

Get the action string associated with this permission

Returns:

the allowed actions separated by commas

6.6.11 implies

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

Determines if the specified permission is implied by this permission.

This method returns false iff any of the following conditions are fulfilled for the specified permission:

- it is not a MonitorPermission
- it has a broader set of actions allowed than this one
- it allows initiating time based monitoring jobs with a lower minimal sampling interval
- the target set of Monitorables is not the same nor a subset of the target set of Monitorables of this permission
- the target set of StatusVariables is not the same nor a subset of the target set of StatusVariables of this permission

Parameters:

p - the permission to be checked

Returns:

true if the given permission is implied by this permission

All Page Within This Box

7 Considered Alternatives

7.1 Combining Monitorables with DMT Plugins

In the early phase of specification it seemed feasible to have a common superinterface for Monitorable services and read only DMT Plugins. However after changing to Monitorable API to closely map the semantics of theTrap Management Object (see 35) it is not feasible any more: the semantics of updating the status variable is different and also the data type of a status variable is different from a node in the DMT.

7.2 Using Metatyping for describing status variables

Using the MetaTyping service for describing Monitorable services is possible but it is seen as an unnecessarily complex way of doing it. 99% of real life status variables will be integers. Also the MetaTyping service does not support the Collection Method of a status variable and as this information is not likely to be needed for other services it is not feasible to extend it.

7.3 Having the Monitorables to control the measurement jobs

The Monitorables might take the responsibility of controlling the periodic updates but it would require difficult housekeeping tasks: each monitorable service would need either timers or separate threads to periodically broadcast the updated values, also they would need to keep track of their listeners. This solution is much more complex than the proposed one.

7.4 Control Unit and Diagnostics

The requirement set for Control Unit and Diagnostics services partly overlaps with the service proposed here. However we did not propose unification of this 3 services for the following reasons:

- The scope of Control Unit and Diagnostics is much broader than the Monitoring service.
- Both Control Unit and Diagnostics rely on the MetaTyping service which the MEG Device Management Workstream decided not to use in the Monitoring service.
- Features like the periodic update of variables and the specific collection methods are unique to the Monitoring service.

7.5 Allowing additional types for Status Variable values

The selection of allowing only integers, floats and strings as StatusVariable types might seem arbitrary but these types represent the majority of cases that occur in practice. The idea of allowing all types that are supported by the OSGi Configuration Admin was dropped as it added unnecessary complications without adding much value.

8 Security Considerations

The following actions are controlled by policy rules.

- Querying status variables. The Monitor Admin can check the rights of the caller. Reading and resetting status variables requires MonitorPermission with the 'read' action.
- Initiating measurement jobs. The Monitor Admin can check the rights of the caller. Creating measurement jobs requires MonitorPermission with the 'startjob' action.
- Publishing status variables. Reading status variables requires MonitorPermission with the 'publish' action. Note that this permission can not be enforced when a Monitorable registers to the framework because the Service Registry does not know about this permission. Instead, the MonitorAdmin implemenatation must make sure that when a status variable is queried, it is shown only if the Monitorable is authorized to publish the given status variable. One possible way for the Admin to check the permissions is to use the framework's `Bundle.hasPermission()` method.
- Switching event sending on and off. So as not to overflow the system with monitoring events the entities having MonitorPermission with the 'switchevents' action present can control whether the change in given StatusVariables fires events.
- Listing Monitorable services and StatusVariables. Callers having MonitorPermission with the 'discover' action can list the Monitorable services currently registered and the StatusVariables a given Monitorable publishes.

Registering the MonitorAdmin and MonitorListener interfaces must be guarded by ServicePermissions.

9 Document Support

9.1 References

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

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

[3]. SyncML Device Management Tree Description

[4]. RFC 0078 MEG High-level Architecture

[5]. RFP 0058 Device Management

All Page Within This Box

[6]. RFP 0055 MEG Policy Framework

[7]. RFP 0062 MEG Glossary

[8]. RFC 0087 DMT Structure

[9]. OSGi R3 specification

[10]. ETSI Performance Management [TS 132 403]

[11]. RFC 0085 Device Management

[12]. RFC 0097 Generic Event Mechnanism

9.2 Author's Address

Name	Balázs Gődény
Company	Nokia
Address	Hungary, 1092 Budapest, Köztelek u. 6.
Voice	+36209849857
e-mail	balazs.godeny@nokia.com

9.3 Acronyms and Abbreviations

See 0.2.

9.4 End of Document

All Page Within This Box