# RFC 92 - MEG Policy Admin

Confidential, Draft

29 Pages

## Abstract

This document describes OSGi Mobile Expert Group Policy Admin interface.

Deleted: 29
Inserted: 29
Deleted: 29

# 0 Document Information

## 0.1 Table of Contents

Deleted: 6
Deleted: 14
Deleted: 20
Deleted: 22
Deleted: 26
Deleted: 26
Deleted: 26
Deleted: 26
Deleted: 27

All Page Within This Box

Deleted: 29
Inserted: 29
Deleted: 29
Deleted: 27
Deleted: 28
Deleted: 28
Deleted: 28
Deleted: 28
Deleted: 29
Deleted: 29

## 0.2 Status

This document specifies Policy Admin interface for the OSGI Alliance, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within the OSGi Alliance.

## 0.3 Acknowledgement

The author would like to acknowledge the work of the MEG Policy workstream participants.

Pavlin Dobrev, ProSyst

BJ Hargrave, IBM

Biju Kaimal, Motorola

Peter Kriens, aQute

Benjamin Reed, IBM

## 0.4 Terminology and Document Conventions

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

`Source code is shown in this typeface.`

## 0.5 Revision History

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

| Revision | Date | Comments |
|---|---|---|
| Initial | July 1 2004 | Initial version<br>Gabor Paller, Nokia, gabor.paller@nokia.com |
| 0.1 | July 7, 2004 | Updated after 2004.07.06 phone conference<br>Gabor Paller, Nokia, gabor.paller@nokia.com |

All Page Within This Box

Deleted: 29

Inserted: 29

Deleted: 29

| Revision | Date | Comments |
|---|---|---|
| 0.2 | July 7, 2004 | Added text for section 5.2 “Management tree structure for Permissions”<br>Biju Kaimal, Motorola, biju@motorola.com |
| 0.3 | July 12, 2004 | Changed policy DMT model to match RFC 73<br>Biju Kaimal, Motorola, biju@motorola.com<br>Added conditional permission sections.<br>Gabor Paller, Nokia, gabor.paller@nokia.com |
| 0.4 | July 22, 2004 | Added fine-grained management permissions section<br>Added some clarification to the conditional permission framework<br>Gabor Paller, Nokia, gabor.paller@nokia.com |
| 0.5 | July 22, 2004 | Changed the DMT policy representation based on input from Ben Reed<br>Biju Kaimal, Motorola. biju@motorola.com |
| 0.6 | Aug. 11, 2004 | Conditional permission section adapted to RFC-95<br>Policy management section added<br>Gabor Paller, Nokia, gabor.paller@nokia.com |
| 0.7 | Aug. 16, 2004 | Changes discussed during the Aug. 13 meeting incorporated.<br>Gabor Paller, Nokia, gabor.paller@nokia.com |
| 0.8 | Sept. 2, 2004 | Changes discussed during the Aug. 24 Boston face-to-face meeting incorporated.<br>Gabor Paller, Nokia, gabor.paller@nokia.com<br>New version of the permission DMT – now the permissions are stored in encoded form<br>Biju Kaimal, Motorola, biju@motorola.com |
| 0.9 | Sept 8, 2004 | Changed DMT based on discussion from conf. call<br>Biju Kaimal, Motorola. biju@motorola.com<br>isEvaluated-isSatisfied swap at TransferCost, UserPrompt, IMSI and IMEI classes in the Javadoc section<br>Gabor Paller, Nokia, gabor.paller@nokia.com |
| 0.91 | Sept 15, 2004 | Added SHA1- calculation order for node name and explanation for why there are different DM subtrees for Java 2 and MIDP permissions.<br>Biju Kaimal, Motorola. biju@motorola.com |

Deleted: 29

Inserted: 29

Deleted: 29

| Revision | Date | Comments |
|---|---|---|
| 0.10 | Oct 26, 2004 | Followed changes in RFC95 (Conditional Permissions), added TransferCost static methods<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.11 | Nov 19, 2004 | Re-work of DMT ./OSGi/Policies/Java structure<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.12 | Nov 29, 2004 | Simplify ./OSGi/Policies/Java structure, clarify ConditionalPermission hash calculation<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.13 | Dec 7, 2004 | Changed hexadecimal encoding to base64, minor clarifications, drawings<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.14 | Dec 9, 2004 | Removed DMTPermission, as it is now in rfc 85<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.15 | Dec 15, 2004 | UserPrompt localization<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.16 | Jan 12, 2005 | % sign in UserPrompt localization<br><br>org.osgi.meg namespace moved to org.osgi.util.gsm and mobile<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.17 | Jan 19, 2005 | Removed MIDP tree<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.18 | Jan 25, 2005 | Removed MEGMgmtPermission (all its functionality moved to other RFCs)<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.19 | Jan 31, 2005 | Clarified how the management tree behaves in writing<br><br>Clarified Java Permission checks on tree<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.20 | Feb 10, 2005 | Changed condition constructors to getInstance static methods<br><br>Peter Nagy <peter.1.nagy@nokia.com> |
| 0.21 | Feb 10, 2005 | Clarification for native message localization<br><br>Clarification of catalog naming<br><br>Peter Nagy <peter.1.nagy@nokia.com> |

Deleted: 29

Inserted: 29

Deleted: 29

| Revision | Date | Comments |
|---|---|---|
| 0.22 | Feb 25, 2005 | Removed out-of date text referring to OSGi/Policies/MgmtServer<br>Peter Nagy <peter.1.nagy@nokia.com> |

# 1 Introduction

MEG has identified the importance of policy and policy management and documented the policy requirements in RFP-55 [4]. Based on the MEG RFPs the MEG high-level architecture (RFC-78) [5] was created that identified the Policy Manager component and its interface. This document is the detailed specification of the Policy Manager and the related mechanisms.

The goal of the MEG Policy workstream is to create a coherent picture of MEG policy. As the activity proceeds, the workstream may decide to delegate some work items to other workstreams.

# 2 Application Domain

Terminology used in this document can be found in MEG Glossary (RFP-62) [3].

Security concerns are always important in case of management systems. This is related to the power of the operations available on the management interface. It is important therefore that the rules, which actor can launch what management operation are described in a clear, flexible and manageable way. The collection of these rules (or permissions) is called policy.

We have multiple actors in the system envisioned by the OSGi MEG activity. Management operations can be launched remotely, from management servers (where different management servers may have different policy associated to them) or locally by applications or end users. We also expect the policy rules to be very different in different usage scenarios depending on the ownership of the device, the mobile operator’s business model, the subscription plan, the corporate security policy, etc

Deleted: 29
Inserted: 29
Deleted: 29

# 3 Problem Description

OSGi MEG envisions a system where the operation of the system can be described flexibly. The policy affects the system in the following ways:

- Policy controls the management system. When an administrator or an application having management rights tries to launch a management operation (either locally or remotely) the policy system is consulted and the management operation is accepted/rejected based on the policy decision.

- Policy controls the running applications. When an application wants to execute a privileged operation, the policy system is consulted and the operation is executed/rejected based on the policy decision.

OSGi itself has a policy system and MEG endorsed OMA DM that has another policy system. The goal of the MEG Policy work is to find the right task distribution between the two policy systems when satisfying MEG policy requirements, enhancing these systems if necessary and introducing solutions that guarantee the correct cooperation of the two policy systems. The MEG high-level architecture [5] is based heavily on OMA DM tree model [6]. DMT has an ACL mechanism that is able to control remote access to the DMT. The DMT (along with DMT ACLs) can locally be manipulated through the interfaces specified in [7]; remote DMT management (including ACLs) is described in the OMA DM protocol specification [8].

Other important mechanism in MEG policy is the Java/OSGi permissions system. The permission system controls access of Java programs to different resources. Permissions are manageable remotely; this document specifies DMT structure for permission management. Policy Admin uses standard OSGi Permission Admin to manipulate the permissions. In order to satisfy several requirements in [4], the Permission Admin also needs to be enhanced. MEG Policy builds on the mechanisms introduced in RFC-73 [9] and RFC-95 [11].

Policy Admin provides policy management features. Access to policy management is also controlled by the policy.

The ACL and the permission system together control the access to MEG management resources and operations. Remote access is controlled by DMT ACLs and optionally, if there are permissions associated with the operation behind the management object, Java permissions provide more fine-grained control. Local access is controlled by Java permissions.

# 4 Requirements

*This section should be copied from the appropriate RFP(s)*

Deleted: 29
Inserted: 29
Deleted: 29

# 5 Technical Solution

## 5.1 Relation of ACL and Permission-based policy systems

As described in section 3, OSGi MEG solution has two existing policy systems. In order to provide the administrator of the system with clear, unified view of the system policy determined by both ACLs and Permission, relation of the two policy systems need to be clarified.

The administrator can access the management system from two directions.

- The administrator connects to the management system remotely, through a remote management protocol.
- The administrator executes the management action locally, invoking a local interface from a Java program.

In case of remote access, the ACLs of the management objects are checked and the management operations are accepted or rejected based on the identity of the remote administrator. Depending on the semantics behind the management object, MEG management functionality is invoked. MEG management functionality may be protected by Java permission to provide more fine-grained policy control. Support of the permission-based fine grain policy check is optional.

If the device supports fine-grained permission-based policy check and the ACLs on the relevant management objects grant access but the fine-grained permission(s) are not set, access is denied. In other words, if the device supports fine-grained management permissions, these permissions have to be set else no access to the protected management operation will be granted.

If the device supports fine-grained permission-based policy checks, then those tree parts that are representing an already existing service Java API, must employ the same security checks. Thus, a management entity (principal) with a given permission set, can only do things that a bundle with the same set of permissions can do with accessing the Java API directly.

Local bundles can also access management objects over the DMT Admin interface [7]. The local access to the DMT is controlled by the DMTPermission permission (seeRFC 85 - Device Management).

## 5.2 Management tree structure for Permissions

The different security policies present in the OSGi MEG system are stored inside the Device Management Tree (DMT), so that they can be managed OTA by the operator or enterprise administrator. This section presents how the different policies in the OSGi MEG environment are represented inside the DMT.

[TODO: It might be helpful to shown a diagram in section 5.1, showing the interactions between the various subsystems like PolicyAdmin, PermissionAdmin, DMTAdmin, DeploymentAdmin, etc]

In the MEG environment, there are OSGi bundles written in Java. In addition, MIDlets might be supported too. There are separate security policies for these execution environments. Also in the case of MEG, where bundle

All Page Within This Box

are deployed on the system, there needs to be policy to determine what privileges should be assigned to the bundle at installation time inside PermissionAdmin.

Figure 5.3.1 shows how the PermissionAdmin policy is represented inside the DMT.

The subtree under ./OSGi/Policies/Java stores the different kinds of permissions associated with different entities that depend on Java 2 security. These are permissions that are managed by PermissionAdmin, ConditionalPermissionAdmin or permissions assigned to Management server ids.

The different nodes in the subtree are:

./OSGi/Policies/Java/Bundle: stores the Java 2 permissions associated with a bundle, based on location. This subtree represents the functionality of the PermissionAdmin service

./OSGi/Policies/Java/ConditionalPermission: stores the conditional permissions given to bundles. This subtree represents the functionality of the ConditionalPermissionAdmin service of rfc-95 [11].

./OSGi/Policies/Java/DmtPrincipal: stores the permissions associated with different DMT principals.

### 5.2.1 PermissionInfo nodes

Permissions in OSGi are described with an array of org.osgi.service.permissionadmin.PermissionInfo objects. The representation of this array in the tree is with a node called PermissionInfo, that has a value of string. The string contains encoded permissioninfos as given by the PermissionInfo.getEncoded() function of the framework. The individual entries are trailed by a newline (\u000a) character.

Example:

`(org.osgi.framework.PackagePermission ”org.osgi.*” ”IMPORT\”)`***newline***

`(org.osgi.framework.ServicePermission ”org.osgi.service.http.*” ”GET”)`***newline***

### 5.2.2 Bundle Permissions

The node ./OSGi/Policies/Java/Bundle contains data that represents the bundle permissions accessible via the service org.osgi.service.permissionadmin.PermissionAdmin.

There is one sub-node ./OSGi/Policies/Java/Bundle/Default that gives access to the default permissions. These are also accessible locally via the PermissionAdmin API setDeafultPermissions(), getDefaultPermissions() function calls. Note that PermissionAdmin differentiates between calling setDefaultPermissions with null and calling it with an empty array. Deleting the Default node is equivalent to calling setDefaultPermissions with null parameter. Creating a Default node with zero permissions is equivalent to calling setDefaultPermissions with an empty PermissionInfo array.

Permissions based on locations are in nodes under ./OSGi/Policies/Java/Bundle/*[location-hash]*. The hash is calculated by the following algorithm:

1. take the UTF-8 encoding of the string without trailing zeroes
2. compute the SHA-1 digest [12]
3. base64-encode the digest
4. remove trailing ‘=’ characters (as they are redundant)

5. replace forward slash '/' characters with underscore '_'

Example: the node name of the permissions associated with location "http://example.com/location1" isfYWPcayGULN1Dkuqd62c5zkdiu4. (Rationale: there is a limit to the length of the DMT node name, so using the location string in the path name directly may not be feasible) Under every node there is a Location node that contains the location string as value. Thus, the node ./OSGi/Policies/Java/Bundle/fYWPcayGULN1Dkuqd62c5zkdiu4/Location contains the string value "http://example.com/location1". There is a node for all locations given by the PermissionAdmin.getLocations() function.

All nodes under ./OSGi/Policies/Java/Bundle/*[location-hash]* and the node ./OSGi/Policies/Java/Bundle/Default contain a node PermissionInfo that lists the permissions associated with the given location.

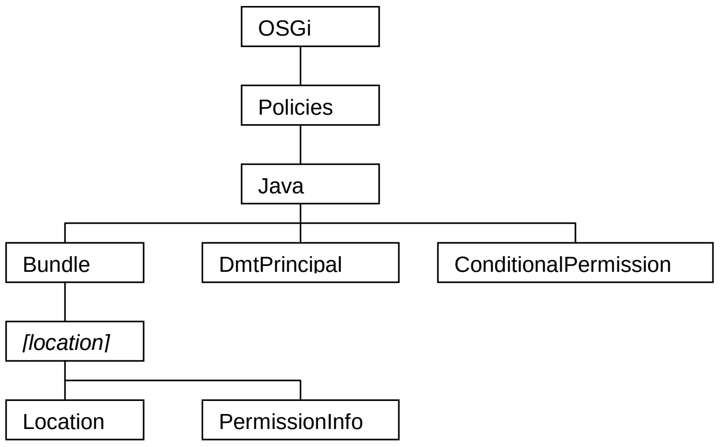
Example, specifying a basic default permission.

./OSGi/Policies/Java/Bundle/Default/PermissionInfo := (org.osgi.framework.PackagePermission * IMPORT)***newline***

Example, specifying adminpermission to location http://example.com/location1:

./OSGi/Policies/Java/Bundle/fYWPcayGULN1Dkuqd62c5zkdiu4/Location := "http://example.com/location1"

./OSGi/Policies/Java/Bundle/fYWPcayGULN1Dkuqd62c5zkdiu4/PermissionInfo :=
(org.osgi.framework.AdminPermission * *)***newline***



### 5.2.3 DMT principals

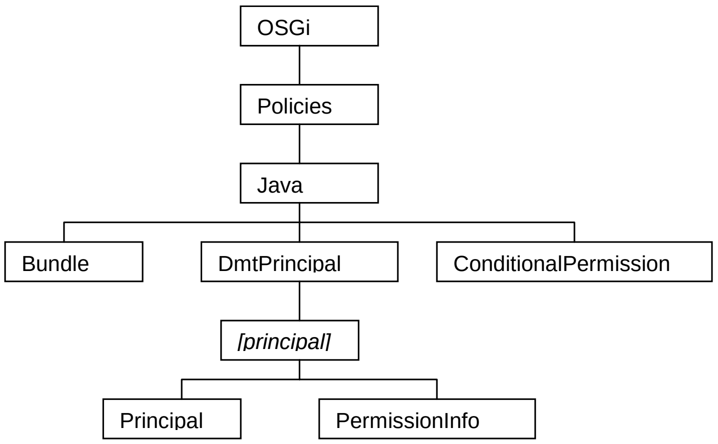
This tree represents permissions associated with different DMT principals. The principal is a string that identifies a manager entity. It is given to the DMT Admin by the different protocol plugins, at Session creation. Based on this string, and what is specified in this subtree, the DMT Admin will assign permissions to the different management servers. See rfc-85 for more details on how this is done.

./OSGi/Policies/Java/DmtPrincipal contains the tree describing the permissions for the different principals.

./OSGi/Policies/Java/DmtPrincipal/*[principal-hash]*/ is a node containing permission data for a principal. The principal-hash is computed by the same string->UTF-8->sha-1->base64 conversion as with the bundle locations in 5.2.2.

./OSGi/Policies/Java/DmtPrincipal/*[principal-hash]*/Principal contains the principal as a string.

./OSGi/Policies/Java/DmtPrincipal/*[principal-hash]*/PermissionInfo contains the permissions associated with this principal. The structure is specified in 5.2.1.



### 5.2.4 Conditional Permissions

The ./OSGi/Policies/Java/ConditionalPermission tree represents the org.osgi.service.condpermadmin ConditionalPermissionAdmin service (see rfc 95 [11])

There is a node for every ConditionalPermissionInfo. It contains a PermissionInfo node as in 5.2.1. It also contains a ConditionInfo node, that contains the conditions.

The ConditionInfo has the same structure as a PermissionInfo, but has the string representation of a condition in every value, as returned by ConditionInfo.getEndoded().

The name of every node is calculated by the following algorithm:

1. Encode the PermissionInfo objects with the getEncoded() function. Put a newline character (\u000a) at the end of every string. Order them lexicographically as specified by the java.lang.String.compareTo(String) function. Concatenate them in this order.

2. Encode the ConditionInfo objects with the getEncoded() function. Put a newline character (\u000a) at the end of every string. Order them lexicographically as specified by the java.lang.String.compareTo(String) function. Concatenate them in this order.

3. Concatenate the two strings with the PermissionInfo string first.

4. Calculate the resulting hash string as in 5.2.2

Example:

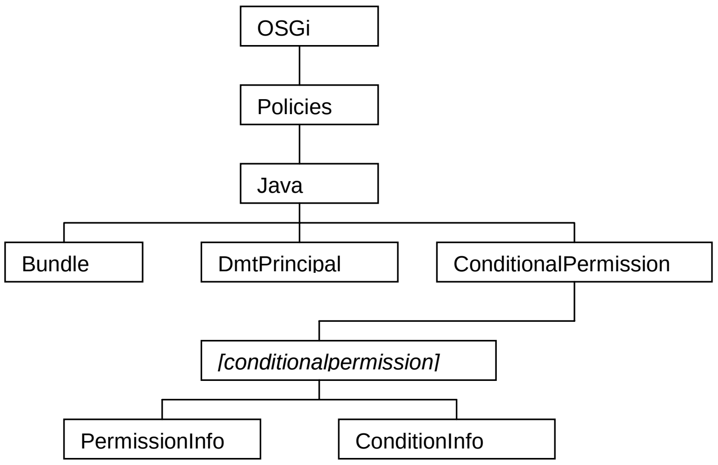
Deleted: 29
Inserted: 29
Deleted: 29

./OSGi/Policies/Java/ConditionalPermission/WovYXjHL_EgRTOVWHgipOk82tt8/

…/ConditionInfo = [org.osgi.service.condpermadmin.BundleLocationCondition "http://example.com/loc1"]***newline***

…/PermissionInfo = (org.osgi.framework.ServicePermission "org.osgi.service.http.HttpService" "register") ***newline***(org.osgi.framework.PackagePermission "org.osgi.*" "IMPORT")***newline***

Note: the hash is valid in this example. Please check if you understood the algorithm right by re-calculating the hash and comparing it with the one in the example.



### 5.2.5 Tree behaviour when modifying

The plugins under ./OSGi/Policies/Java subtree MUST support atomic sessions.When modifying the subtree, the DMT Session must be opened to atomic (see org.osgi.service.dmt.DmtSession.LOCK_TYPE_ATOMIC in RFC 85 - Device Management). This ensures that in case of communication failure, the security settings are not left in an intermediate state – the plugin must roll back to the original state.

If at the end of an atomic session the tree is left in an inconsistent state (like, there is a null Location entry somewhere, or the PermissionInfo data is not valid), the whole session MUST be rolled back to the original state.

It is not needed for the remote management entity to calculate the hashes when creating a new entry in the tree. Any name could be used. These ad-hoc names will remain ‘till the end of the atomic session. After a successful session close, they will be renamed to the calculated hashes.Example session:

- open (ATOMIC)
- create ./OSGi/Policies/Java/1
- set value ./OSGi/Policies/Java/1/Location = *[location]*
- set value ./OSGi/Policies/Java/1/PermissionInfo = *[permissioninfos]*
- close

All Page Within This Box

After this, there will be a new location entry in the Permission Admin. If someone else opens the ./OSGi/Policies/Java subtree, there won't be an "1" node anymore, but one with the properly calculated hash.

## 5.3 Fine-grained management permissions

### 5.3.1 Relationship of management permissions to management servers and bundles

MEG policy system protects both the application run-time behavior and the access to the management functionality. In this section the management system access control is described.

The MEG policy system builds on the interaction of OMA DM ACL and Java permission system. The usage of ACLs and permissions depends on the direction the management system is approached from.

- If the management system is contacted remotely, by means of OMA DM protocol, OMA DM Management Agent always checks the ACLs first. If the ACL permits access for the given management server, the management agent eventually calls the management object glue code which invokes the appropriate management agent function according to the management object's semantics. The call to the OSGi management code may be protected by fine-grained management permissions.
- The management agent functionality may be accessible over Java interface as well providing local access to management functions. In this case the Java interfaces are protected by the fine-grained management permissions.

According to the use cases, the management permissions can be associated to two different entities.

- If the management permissions are associated to management servers, the management agent uses these permissions to limit its own permissions by e.g. AccessController.doPrivileged call with an AccessControlContext parameter using the permissions of the management server. The permissions of each management server are mapped to the DMT. The DMT root for management server permissions is ./OSGi/Policies/DmtPrincipal. See 5.2.3 for the exact structure.
- Local bundles having management rights may own management permissions like any other permission.

**Deleted:** MgmtServer/[serverid]. Under each serverid node there are a set of PermissionInfo branches. The PermissionInfo branches conform to the node structure described in the "Management tree structure for Permissions" section. *This text is just a placeholder to indicate that permissions associated to management servers can be managed through the DMT. The exact DMT will be specified in section 5.2*

### 5.3.2 Management permissions by RFP-55 functionality group

#### *5.3.2.1 Install, uninstall and update*

AdminPermission( "<filter>","lifecycle" ) allows fine-grained control of these operations. Filtering is possible by location, signer or bundle symbolic name.

This permission may be combined by user confirmation or transfer cost using the conditional permission framework.

Deleted: 29
Inserted: 29
Deleted: 29

*Permissions related to UDLs are subject to Deployment track progress.*

### *5.3.2.2 Inventory query*

MEGMgmtPermission( “<filter>”,”inventory” ) allows fine-grained control over who can query the detailed management agent inventory. The filter string is specified similarly to AdminPermission() in RFC-73..

### *5.3.2.3 Configuration*

AdminPermission( “<filter>”,”configuration” ) allows fine-grained control over accessing the configuration set of the bundles matching the filter.

*There’s a discrepancy here between SyncML DM functionality that allows per-parameter and per-access mode (read/write/create) access control and AdminPermission( “configuration” ) that allows only total access to the configuration sets of the bundles matching the filter.*

User confirmation is handled through the conditional permission framework.

### *5.3.2.4 KPI management*

MEGMgmtPermission( “<filter>”,”KPIpublish” ) allows fine-grained control that the bundle is allowed to publish the Monitorable ID and KPI ID matching the filter string. Filter string is a simple string that may continue wildcards. This permission is used to protect only local access.

KPI read access is provided by the DMT and therefore the KPI read access control is provided by DMTPermission.

MEGMgmtPermission( “<filter>”,”KPIperiodical” ) allows fine-grained control over whether periodical KPI update is allowed for the holder of the permission. Holder of this permission is allowed to launch a periodical KPI update jobs. <Filter> is an OSGi filter as defined by org.osgi.framework.Filter. Fields of the filter can be period. Period is the minimum sampling time between samples (in milliseconds) while reports is the maximum length of the monitoring job in minutes.

### *5.3.2.5 Log access*

*New permission needs to be created that protects minimally the Log Reader facility. It should be possible to say that this LogReader can receive only log entries from certain bundles.*

### *5.3.2.6 Execution state*

AdminPermission( “<filter>”,”execute” ) protects the start/stop actions on bundles as described in RFC-73.

Application Model workstream will define the permission mechanism for protecting the lifecycle changes of UEs.

Deleted: 29
Inserted: 29
Deleted: 29

The JVM/OSGi execution state change actions are only protected by ACLs.

#### *5.3.2.7 Integrity check*

MEGMgmtPermission( "<filter>","integrity" ) allows fine-grained control over who can launch integrity check over what other bundles. The holder of the permission can launch integrity check on the bundles that match the RFC-73-style filter string.

#### *5.3.2.8 Operator- and device characteristic-specific requirements*

The conditional permission framework allows attaching conditions like IMSI and IMEI to any permission.

## 5.4 Policy management

MEG policy system consists of two independent elements. These are managed separately.

- The DMT ACL system is managed remotely through the OMA DM protocol or locally over DMT Admin. Bundle-related management objects are managed through their associated mechanisms (e.g. configuration management, monitoring) therefore there is no need for bundles to programmatically create/destroy management objects.

- Java permissions control access to many parts of the system. Java permissions can be managed remotely by the DMT or locally by the appropriate Permission Admin and Conditional Permission Admin interfaces.

## 5.5 Java API

### 5.5.1 DMTPermission

The org.osgi.service.dmt.DMTPermission controls access to the DM tree. It is specified in RFC 85.

### 5.5.2 UserPrompt

org.osgi.util.mobile
Class UserPromptCondition

java.lang.Object
└ **org.osgi.util.mobile.UserPromptCondition**

**All Implemented Interfaces:**
org.osgi.service.condpermadmin.Condition

public class **UserPromptCondition**
extends java.lang.Object
implements org.osgi.service.condpermadmin.Condition

Class representing a user prompt condition. Instances of this class hold two values: a prompt string that is to be displayed to the user and the permission level string according to MIDP2.0 (oneshot, session, blanket).

## Constructor Summary

All Page Within This Box

Deleted: 29
Inserted: 29
Deleted: 29

| |
|---|
| **UserPromptCondition**() |

## Method Summary

| | |
|---|---|
| static org.osgi.service.condpermadmin.Condition | **getInstance**(org.osgi.framework.Bundle bundle, java.lang.String levels, java.lang.String defaultLevel, java.lang.String catalogName, java.lang.String message)<br>Creates an UserPrompt object with the given prompt string and permission level. |
| boolean | **isEvaluated**()<br>Checks whether the condition is evaluated. |
| boolean | **isMutable**()<br>Checks whether the condition may change in the future. |
| boolean | **isSatisfied**()<br>Checks whether the condition is satisfied. |
| boolean | **isSatisfied**(org.osgi.service.condpermadmin.Condition[] conds, java.util.Dictionary context)<br>Checks an array of UserPrompt conditions |

| **Methods inherited from class java.lang.Object** |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

## Constructor Detail

### *5.5.2.1 UserPromptCondition*

public **UserPromptCondition**()

## Method Detail

### *5.5.2.2 getInstance*

```
public static org.osgi.service.condpermadmin.Condition getInstance(org.osgi.framework.Bundle bundle,
                                                     java.lang.String levels,
                                                     java.lang.String defaultLevel,
                                                     java.lang.String catalogName,
                                                     java.lang.String message)
```

Creates an UserPrompt object with the given prompt string and permission level. The user should be given choice as to what level of permission is given. Thus, the lifetime of the permission is controlled by the user.

**Parameters:**

bundle - the bundle to ask about
levels - the possible permission levels. This is a comma-separated list that can contain following strings: ONESHOT SESSION BLANKET.
defaultLevel - the default permission level. If it is an empty string, then there is no default.
catalogName - the message catalog base name

All Page Within This Box

Deleted: 29

Inserted: 29

Deleted: 29

message - textual description of the condition, to be displayed to the user. If it starts with a '%' sign, then the message is looked up from the catalog specified by the catalogName parameter. The key is the rest of the string after the '%' sign.

---

#### *5.5.2.3 isEvaluated*

public boolean **isEvaluated**()

Checks whether the condition is evaluated. Depending on the permission level, the function returns the following values.

- ONESHOT - isSatisfied always returns false. The condition reevaluated each time it is checked.
- SESSION - isSatisfied returns false until isEvaluated() returns true (the user accepts the prompt for this execution of the bundle). Then isSatisfied() returns true until the bundle is stopped.
- BLANKET - isSatisfied returns false until isEvaluated() returns true (the user accepts the prompt for the lifetime of the bundle). Then isSatisfied() returns true until the bundle is uninstalled.

**Specified by:**
isEvaluated in interface org.osgi.service.condpermadmin.Condition
**Returns:**
true if the condition is satisfied.

---

#### *5.5.2.4 isMutable*

public boolean **isMutable**()

Checks whether the condition may change in the future.
**Specified by:**
isMutable in interface org.osgi.service.condpermadmin.Condition
**Returns:**

- false, if it is already evaluated and is a BLANKET condition
- true otherwise

---

#### *5.5.2.5 isSatisfied*

public boolean **isSatisfied**()

Checks whether the condition is satisfied. Displays the prompt string to the user and returns true if the user presses "accept". Depending on the amount of levels the condition is assigned to, the prompt may have multiple "accept" buttons and one of them can be selected by default (see deflevel parameter at UserPrompt constructor). It must be always possible to deny the permission (e.g. by a separate "deny" button).
**Specified by:**
isSatisfied in interface org.osgi.service.condpermadmin.Condition
**Returns:**
true if the user accepts the prompt (or accepts any prompt in case there are multiple permission levels).

---

#### *5.5.2.6 isSatisfied*

public boolean **isSatisfied**(org.osgi.service.condpermadmin.Condition[] conds,
java.util.Dictionary context)

Checks an array of UserPrompt conditions
**Specified by:**
isSatisfied in interface org.osgi.service.condpermadmin.Condition

**Parameters:**
conds - the array containing the UserPrompt conditions to evaluate
context - storage area for one-shot evaluation
**Returns:**
true, if all conditions are satisfied

### 5.5.3 IMSI

org.osgi.util.gsm
Class IMSICondition

java.lang.Object
  └ **org.osgi.util.gsm.IMSICondition**
**All Implemented Interfaces:**
org.osgi.service.condpermadmin.Condition

---

public class **IMSICondition**
extends java.lang.Object
implements org.osgi.service.condpermadmin.Condition
Class representing an IMSI condition. Instances of this class contain a string value that is matched against the IMSI of the subscriber.

---

## Method Summary

| | |
|---|---|
| static org.osgi.service.condpermadmin.Condition | **getInstance**(org.osgi.framework.Bundle bundle, java.lang.String imsi)<br>Creates an IMSI condition object |
| boolean | **isEvaluated**()<br>Checks whether the condition is evaluated |
| boolean | **isMutable**()<br>Checks whether the condition can change |
| boolean | **isSatisfied**()<br>Checks whether the condition is true. |
| boolean | **isSatisfied**(org.osgi.service.condpermadmin.Condition[] conds, java.util.Dictionary context)<br>Checks whether an array of IMSI conditions match |

| **Methods inherited from class java.lang.Object** |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

## Method Detail

#### *5.5.3.1 getInstance*

public static org.osgi.service.condpermadmin.Condition **getInstance**(org.osgi.framework.Bundle bundle,
java.lang.String imsi)

Creates an IMSI condition object
**Parameters:**
bundle - ignored
imsi - The IMSI value of the subscriber.

### *5.5.3.2 isSatisfied*

public boolean **isSatisfied**()

Checks whether the condition is true. The IMSI of the object instance is compared against the IMSI of the device's user.

**Specified by:**

isSatisfied in interface org.osgi.service.condpermadmin.Condition

**Returns:**

true if the IMSI value match.

### *5.5.3.3 isEvaluated*

public boolean **isEvaluated**()

Checks whether the condition is evaluated

**Specified by:**

isEvaluated in interface org.osgi.service.condpermadmin.Condition

**Returns:**

always true

### *5.5.3.4 isMutable*

public boolean **isMutable**()

Checks whether the condition can change

**Specified by:**

isMutable in interface org.osgi.service.condpermadmin.Condition

**Returns:**

always false

### *5.5.3.5 isSatisfied*

public boolean **isSatisfied**(org.osgi.service.condpermadmin.Condition[] conds,
java.util.Dictionary context)

Checks whether an array of IMSI conditions match

**Specified by:**

isSatisfied in interface org.osgi.service.condpermadmin.Condition

**Parameters:**

conds - an array, containing only IMSI conditions

context - ignored

**Returns:**

true, if they all match

## 5.5.4 IMEI

org.osgi.util.gsm

Class IMEICondition

java.lang.Object
└ **org.osgi.util.gsm.IMEICondition**

**All Implemented Interfaces:**

org.osgi.service.condpermadmin.Condition

public class **IMEICondition**

All Page Within This Box

Deleted: 29
Inserted: 29
Deleted: 29

extends java.lang.Object
implements org.osgi.service.condpermadmin.Condition
Class representing an IMEI condition. Instances of this class contain a string value that is matched against the IMEI of the device.

## Method Summary

| | |
|---|---|
| static org.osgi.service.condpermadmin.Condition | **getInstance**(org.osgi.framework.Bundle bundle, java.lang.String imei) Creates an IMEI object. |
| boolean | **isEvaluated**() Checks whether the condition is evaluated. |
| boolean | **isMutable**() check whether the condition can change. |
| boolean | **isSatisfied**() Checks whether the condition is satisfied. |
| boolean | **isSatisfied**(org.osgi.service.condpermadmin.Condition[] conds, java.util.Dictionary context) Checks for an array of IMEI conditions if they all match this device |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

## Method Detail

### *5.5.4.1 getInstance*

public static org.osgi.service.condpermadmin.Condition **getInstance**(org.osgi.framework.Bundle bundle, java.lang.String imei)

Creates an IMEI object.
**Parameters:**
bundle - ignored
imei - The IMEI value of the device.

### *5.5.4.2 isSatisfied*

public boolean **isSatisfied**()

Checks whether the condition is satisfied. The IMEI of the object instance is compared against the IMEI of the device.
**Specified by:**
isSatisfied in interface org.osgi.service.condpermadmin.Condition
**Returns:**
true if the IMEI value matches.

### *5.5.4.3 isEvaluated*

public boolean **isEvaluated**()

Checks whether the condition is evaluated.

All Page Within This Box

**Specified by:**
isEvaluated in interface org.osgi.service.condpermadmin.Condition
**Returns:**
if the condition status can be evaluated instantly

---

### *5.5.4.4 isMutable*

public boolean **isMutable**()
check whether the condition can change.
**Specified by:**
isMutable in interface org.osgi.service.condpermadmin.Condition
**Returns:**
always false

---

### *5.5.4.5 isSatisfied*

public boolean **isSatisfied**(org.osgi.service.condpermadmin.Condition[] conds,
java.util.Dictionary context)
Checks for an array of IMEI conditions if they all match this device
**Specified by:**
isSatisfied in interface org.osgi.service.condpermadmin.Condition
**Parameters:**
conds - an array, containing only IMEI conditions
context - ignored
**Returns:**
true if all conditions match

### 5.5.5 TransferCost

org.osgi.util.mobile
Class TransferCostCondition

java.lang.Object
└ **org.osgi.util.mobile.TransferCostCondition**
**All Implemented Interfaces:**
org.osgi.service.condpermadmin.Condition

---

public class **TransferCostCondition**
extends java.lang.Object
implements org.osgi.service.condpermadmin.Condition
Class representing abstract transfer cost values. Implementation assigns concrete bearers to abstract transfer costs.
There is a static setTransferCost() method that sets the current transfer cost in a thread local variable. The TransferCost condition will check this, and will fail if the cost is greater. It is recommended that resetTransferCost() is always put in a finally block, to ensure that if something fails, the cost value is removed.

---

| **Field Summary** | |
|---|---|
| static java.lang.String | **HIGH** |

| | |
|---|---|
| static java.lang.String | **LOW** |
| static java.lang.String | **MEDIUM** |

## Method Summary

| | |
|---|---|
| static org.osgi.service.condpermadmin.Condition | **getInstance**(org.osgi.framework.Bundle bundle, java.lang.String costLimit) Creates a TransferCostCondition object. |
| boolean | **isEvaluated**() Checks whether the condition is evaluated. |
| boolean | **isMutable**() check whether the condition can change. |
| boolean | **isSatisfied**() Checks whether the condition is satisfied. |
| boolean | **isSatisfied**(org.osgi.service.condpermadmin.Condition[] conds, java.util.Dictionary context) Checks for an array of TransferCost conditions if they all match |
| static void | **resetTransferCost**() Resets the transfer cost. |
| static void | **setTransferCost**(java.lang.String cost) Sets a thread-local transfer cost. |

| Methods inherited from class java.lang.Object |
|---|
| equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait |

## Field Detail

### *5.5.5.1 LOW*

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

**See Also:**
Constant Field Values

### *5.5.5.2 MEDIUM*

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

**See Also:**
Constant Field Values

### *5.5.5.3 HIGH*

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

All Page Within This Box

Deleted: 29
Inserted: 29
Deleted: 29

**See Also:**

Constant Field Values

## Method Detail

### 5.5.5.4 getInstance

public static org.osgi.service.condpermadmin.Condition **getInstance**(org.osgi.framework.Bundle bundle,
java.lang.String costLimit)

Creates a TransferCostCondition object. This constructor is intended to be called when Permission Admin initializes the object.

**Parameters:**

bundle - ignored

costLimit - The abstract limit cost. Possible values are "LOW","MEDIUM" and "HIGH".

### 5.5.5.5 setTransferCost

public static void **setTransferCost**(java.lang.String cost)

Sets a thread-local transfer cost. All isSatisfied() method calls in this thread will check for this cost, and only those permissions will be activated, that have a cost limit higher than this. The caller MUST call resetTransferCost(), after the permission checks are done. If this function is not called, the default behavior is 'no checks permformed, all tests succeed'.

**Parameters:**

cost - the cost of the current transaction. Only those conditions will evaluate to true, that are equal or higher than this.

### 5.5.5.6 resetTransferCost

public static void **resetTransferCost**()

Resets the transfer cost. After this, the transfer cost checks always succeed.

### 5.5.5.7 isSatisfied

public boolean **isSatisfied**()

Checks whether the condition is satisfied. The limit cost represented by this object instance is compared against the cost in the thread context.

**Specified by:**

isSatisfied in interface org.osgi.service.condpermadmin.Condition

**Returns:**

true if the cost in this condition is greater or equals the cost limit set in thread context

### 5.5.5.8 isEvaluated

public boolean **isEvaluated**()

Checks whether the condition is evaluated.

**Specified by:**

isEvaluated in interface org.osgi.service.condpermadmin.Condition

**Returns:**

true if the condition status can be evaluated instantly

### 5.5.5.9 isMutable

public boolean **isMutable**()

check whether the condition can change.
**Specified by:**
isMutable in interface org.osgi.service.condpermadmin.Condition
**Returns:**
true if the condition can change

### *5.5.5.10 isSatisfied*

public boolean **isSatisfied**(org.osgi.service.condpermadmin.Condition[] conds,
java.util.Dictionary context)

Checks for an array of TransferCost conditions if they all match
**Specified by:**
isSatisfied in interface org.osgi.service.condpermadmin.Condition
**Parameters:**
conds - an array, containing only TransferCost conditions
context - ignored
**Returns:**
true if all transfer costs are below limit

## 5.6 Localization

User-visible strings that need localization only appear in the UserPrompt condition (5.5.2).

### 5.6.1 Loading messages

Every message is described by a catalog name and a key. The catalog name and the locale together specify a properties resource file that contains the translations for the given language. The resource files are loaded via the java classLoader mechanism. The resource files are searched in this order:

- catalogName+"_"+language+"_"+country+".properties"
- catalogName+"_"+language+".properties"
- catalogName+".properties"

The properties file contains messagekey-messagestring mappings, in the java property file format.

Example: if the catalogName is com.provider.messages.userprompt, and the current locale is de-CH, then it will be searched in this order:

- com.provider.messages.userprompt_de_CH.properties
- com.provider.messages.userprompt_de.properties
- com.provider.messages.userprompt.properties

### 5.6.2 ResourceBundle

The algorithm above is similar to java.util.ResourceBundle class behavior, only much more simplified. If the underlying Java2 implementation has ResourceBundle class available, the UserPrompt implementor may choose to use it. In this case the message string translation looks like this:

```
ResourceBundle.getBundle(catalogName).getString(key)
```

### 5.6.3 Message catalog maintenance

Message catalogs can be exported by a bundle that has the necessary export permissions. To continue with the example above, a bundle can export com.provider.messages, and have the files with the translations in its classpath. Thus a remote administrator can install and update message catalogs just like it installs and updates bundles.

To avoid clashes, the message catalog names MUST follow the java package naming conventions (with names like “com.nokia.messages”, “com.somebank.netbankapp”, “net.operatorfoo.messages”, etc.)

### 5.6.4 System native localization

# 6 The OSGi implementer may use specific message catalog names to refer to native (non-Java) localization databases. This specification does not any requirement on how this is implemented. Putting resource bundles with the given name into the class load path via export, may or may not work in this case.Considered Alternatives

It was analyzed whether it is possible to provide automatic synchronization between fine-grained management permissions and ACLs. This approach was considered too problematic; hence it is the administrator’s responsibility to keep ACLs and management permissions in sync.

It was proposed to include the permission condition into the target or action field of a normal Java permission. This solution was not favored because it doesn’t follow the style of the existing Java permissions and conditions could not be associated to existing Java permissions without redefining the syntax of the permission’s target or action field.

In an earlier proposal the conditional permissions were processed entirely in the MEG domain, the framework provided only a hook to the permission logic. This solution couldn’t provide MIDP 2.0-style conditional permission processing where the conditions are present only in the policy.

Deleted: 29

Inserted: 29

Deleted: 29

Currently every management-related permission is collected under the MEGMgmtPermission umbrella. Depending on the progress of other workstreams, the MEGMgmtPermission may be split into multiple permissions. The design in this document tries to minimize the number of permission classes, this is the reason of collecting many functionalities int MEGMgmtPermission.

# 7 Security Considerations

*Description of all known vulnerabilities this may either introduce or address as well as scenarios of how the weaknesses could be circumvented.*

# 8 Document Support

## 8.1 References

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

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

[3]. RFP 62 - MEG Glossary

[4]. RFP 55 – MEG Policy

[5]. RFC 78 – MEG high-level architecture

[6]. SyncML Device Management Tree and Description, http://www.openmobilealliance.org/tech/affiliates/syncml/syncmlindex.html

[7]. RFC 85 - Device Management

[8]. SyncML Device Management Protocol, http://www.openmobilealliance.org/tech/affiliates/syncml/syncmlindex.html

[9]. RFC 73 - Permission Update

[10]. JSR 118: Mobile Information Device Profile 2.0, http://www.jcp.org/en/jsr/detail?id=118

[11]. RFC-95 - Conditional Permissions

All Page Within This Box

Deleted: 29
Inserted: 29
Deleted: 29

[12]. FIPS PUB 180-1: Secure Hash Standard

## 8.2 Author's Address

| | |
|---|---|
| Name | Gabor Paller |
| Company | Nokia |
| Address | Budapest, Köztelek str. 6, 1092, Hungary |
| Voice | +36209369597 |
| e-mail | Gabor.paller@nokia.com |

| | |
|---|---|
| Name | Peter Nagy |
| Company | Nokia |
| Address | Budapest, Köztelek str. 6, 1092, Hungary |
| Voice | +36209849076 |
| e-mail | peter.1.nagy@nokia.com |

## 8.3 Acronyms and Abbreviations

ACL – Access Control List

DM – Device Management

DMT – Device Management Tree

KPI – Key Performance Indicator

MEG – Mobile Expert Group

OMA – Open Mobile Alliance, http://www.openmobilealliance.org/

## 8.4 End of Document