RFC 17: PermissionAdmin Service

Members Only, Final

13 Pages

Abstract

The OSGi framework is in charge of assigning to the bundles running on it the Java 2 permissions they were granted. This RFC defines a service which provides admin−type bundles with the ability to manage these permissions. This service allows for permissions to be setup before a bundle is installed, and for those permissions to be modified at any time during the bundle's lifecycle.

Part of this RFC also defines a synchronous version of the BundleListener. This provides admin−type bundles with hooks into the bundle lifecycle in order to provide "just in time" permission management. It should be noted that the SynchronousBundleListener may also be useful outside the scope of permission management.

0 Document Information

0.1 Table of Contents

0.2 Status

This document specifies the PermissionAdmin interface for the core platform of the Open Services Gateway Initiative, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within OSGi.

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

`Source code is shown in this typeface`.

0.4 Revision History

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

Revision	Date	Comments
Initial	3/13/01	Benjamin Reed, Core Platform Expert Group, OSGi.
Second draft	3/30/01	Minor changes to the Javadoc. Also made PermissionInfo Serializable.
Final 1.0	4/10/01	Removed Serializable and added getEncoded(). Minor wording changes.
Final 1.0	4/13/01	Changed getEncoded to refer to StreamTokenizer instead of StringTokenizer.
Final 1.0	4/18/01	Removed reference to StreamTokenizer. Added equals() and hashCode() to PermissionInfo. Removed comment about actions being comma−separated.
Final 1.0	4/26/01	Changed API comments to allow name to be null.
Final 1.0	4/27/01	Changed encoding format to escape \r and \n.

1 Introduction

Java 2 permissions provide the basis for security in bundle−to−framework and bundle−to−bundle interactions. In version 1.0 of the OSGi Framework Specification it was left up to framework implementers as to how permissions were assigned to bundles. This omission made it impossible to design administrative bundles that would be able to manage bundle permissions in a portable way across different framework implementations.

The PermissionAdmin service provides a standard interface for assigning permissions to bundles based on the bundle location. This allows for permissions to be setup before a bundle is installed, and for the permissions to be modified at any time during the bundle’s lifecycle. Updates to the permissions become effective immediately and persist across restarts of the framework.

If a bundle is installed that does not have permissions assigned to its location, a default set of permissions will be used. The PermissionAdmin service provides a way to set this default set of permissions.

To allow for "just−in−time" permission management, the SynchronousBundleListener interface is defined. This interface extends the BundleListener interface and causes BundleEvents to be delivered synchronously. This allows admin bundles to assign permissions to bundles as they are being installed or updated, that is, "just−in−time", rather than requiring the admin bundles to somehow know in advance if a bundle will be installed.

2 Technical Discussion

The framework MAY provide the PermissionAdmin service. If it does provide this service, there MUST be only one instance of it.

Permissions are assigned to a bundle using the bundle's location string as the key into a table of permissions. The associated value is a collection of permissions. The methods on PermissionAdmin allow this table to be queried and manipulated. Since it is unlikely that the Bundle using the PermissionAdmin is able to resolve all permissions that are to be assigned, and due to the restriction implemented by the framework that only `Permission` classes on the system classpath or from an exported package are to be used for permission checks, the methods on `PermissionAdmin` use arrays of `PermissionInfo` objects (instead of Java 2 `PermissionCollection` objects) to represent sets of permissions.

Since the `PermissionAdmin` must be able to instantiate and persist the permissions it manages, any `Permission` classes referenced in any of the PermissionAdmin's setter methods must have a constructor that takes two arguments: a `name` string and an `actions` string. This is the same restriction imposed by Java 2 on any `Permission` classes referenced in the policy file.

A `PermissionAdmin` may augment the permissions assigned through its setter methods with permissions obtained from another source, such as a Java 2 policy file. The way this is done is implementation dependent.

2.1 Default permissions

The purpose of the default permissions is to specify a set of permissions that is assigned to any bundle that was not explicitly granted any permissions, that is, any bundle that is not represented in the permission table. This default set of permissions is set/gotten using set/getDefaultPermissions, respectively. If there is no default set of permissions, an implementation specific set of permissions may be used as the default set of permissions.

2.2 Permission evaluation

When checking if the permissions assigned to a bundle imply a given target permission, the framework first looks up the bundle location in the table of permissions. If the location is present in the table, the framework will use its associated set of permissions for the permission check. Otherwise, the framework will use the default set of permissions.

For performance reasons, framework implementers will probably not do a table lookup for every permission check. However, frameworks MUST use a bundle's most recent permissions for every permission check. The new permissions may be the result of:

- updating the set of permissions assigned to the bundle location.
- adding or removing a table entry corresponding to the bundle's location, causing the default set of permissions to be no longer used or become applicable, respectively.
- changing the default set of permissions if there is no table entry assigned to the bundle location.

2.3 Just−in−time permission management

In some deployments of the framework it may be possible for the bundle administering the permissions to know ahead of time that a bundle will be installed and setup the necessary permissions. However, one of the goals of this RFC is to also enable more dynamic deployments where bundles are installed without the foreknowledge of the bundle administering the permissions. This is addressed by the `org.osgi.framework.SynchronousBundleListener` interface, which extends `org.osgi.framework.BundleListener`.

The only difference between the two interfaces is the way bundle events are delivered to them: while a `SynchronousBundleListener` receives any bundle events synchronously, a `BundleListener` receives the same events asynchronously. In addition, the framework will call the SynchronousBundleListeners first. A `SynchronousBundleListener` is registered in the same way as a `BundleListener`, that is, using `BundleContext.addBundleListener`. Adding and removing a `SynchronousBundleListener` requires the `AdminPermission`.

Therefore, a bundle administering bundle permissions can register a `SynchronousBundleListener` and wait for `BundleEvents` of type INSTALLED. Upon receiving this event, the admin bundle can lookup the permissions associated with the bundle in an implementation specific way and assign the permissions to the bundle before it is resolved and started.

3 Security Considerations

The `PermissionAdmin` service is a key to controlling the security of the entire framework. Thus, it is extremely important to limit access to it. The PermissionAdmin service is protected by the corresponding `ServicePermission`. In addition, any of its setter methods require `AdminPermission`. This service is always registered by the framework's system bundle and cannot be registered by ordinary bundles.

The ability to access `PermissionAdmin` only gives a Bundle the ability to interrogate permissions unless the Bundle also possesses the `AdminPermission`. A Bundle possessing both the `ServicePermission` to get the `PermissionAdmin` and the `AdminPermission` could grant itself `AllPermission` and disrupt other bundles by removing their permissions.

4 Document Support

4.1 References

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

[2]. Gong, Li, Inside Java 2 Platform Security, Addison Wesley, June 1999.

4.2 Author’s Address

Name	Jan Luehe
Company	Sun Microsystems, Inc.
Address	901 San Antonio Road, UCUP01−207 Palo Alto, CA 94303 USA
Voice	+1 408 863−3216
e−mail	Jan.Luehe@sun.com

Name	Benjamin Reed
Company	IBM
Address	650 Harry Rd, San Jose, CA 95120
Voice	+1 408 927−1811
e−mail	breed@almaden.ibm.com

4.3 org.osgi.service.permissionadmin Interface PermissionAdmin

public interface **PermissionAdmin**

The `PermissionAdmin` service allows gateway administrators to manage the permissions of bundles. There is at most one `PermissionAdmin` service present in the framework.

Access to the `PermissionAdmin` service is protected by corresponding `ServicePermission`. In addition the `AdminPermission` is required to set permissions.

Bundle permissions are managed using a permission table. A bundle’s location serves as the key into the permission table. The value of a table entry is the set of permissions (of type PermissionInfo) granted to the bundle with the given location. A bundle may have an entry in the permission table prior to being installed in the framework.

The permissions specified in `setDefaultPermissions` are used as the default permissions which are granted to all bundles that do not have an entry in the permission table.

Any changes to a bundle’s permissions in the permission table will take effect no later than when bundle’s `java.security.ProtectionDomain` is involved in a permission check, and will be made persistent.

Only permission classes on the system classpath or from an exported package are considered during a permission check. Additionally, only permission classes that are subclasses of `java.security.Permission` and define a 2−argument constructor that takes a *name* string and an *actions* string can be used.

Permissions implicitly granted by the framework (for example, a bundle's permission to access its persistent storage area) cannot be changed, and are not reflected in the permissions returned by `getPermissions` and `getDefaultPermissions`.

Method Summary

`PermissionInfo[]`	**`getDefaultPermissions`**`()` Gets the default permissions.
`java.lang.String[]`	**`getLocations`**`()` Returns the bundle locations that have permissions assigned to them, that is, bundle locations for which an entry exists in the permission table.
`PermissionInfo[]`	**`getPermissions`**`(java.lang.String location)` Gets the permissions assigned to the bundle with the specified location.
`void`	**`setDefaultPermissions`**`(PermissionInfo[] permissions)` Sets the default permissions.
`void`	**`setPermissions`**`(java.lang.String location, PermissionInfo[] perms)` Assigns the specified permissions to the bundle with the specified location.

Method Detail

getPermissions

```
public PermissionInfo[] getPermissions(java.lang.String location)
```

Gets the permissions assigned to the bundle with the specified location.

Parameters:

`location` − The location of the bundle whose permissions are to be returned.

Returns:

The permissions assigned to the bundle with the specified location, or `null` if that bundle has not been assigned any permissions.

---

setPermissions

```
public void setPermissions(java.lang.String location,
                           PermissionInfo[] perms)
```

Assigns the specified permissions to the bundle with the specified location.

Parameters:

`location` − The location of the bundle that will be assigned the permissions.
`perms` − The permissions to be assigned, or `null` if the specified location is to be removed from the permission table.

Throws:

`java.lang.SecurityException` − if the caller does not have the `AdminPermission`.

---

getLocations

`public java.lang.String[] **getLocations**()`

Returns the bundle locations that have permissions assigned to them, that is, bundle locations for which an entry exists in the permission table.

Returns:

The locations of bundles that have been assigned any permissions, or `null` if the permission table is empty.

---

getDefaultPermissions

`public PermissionInfo[] **getDefaultPermissions**()`

Gets the default permissions.

These are the permissions granted to any bundle that does not have permissions assigned to its location.

Returns:

The default permissions, or `null` if default permissions have not been defined.

---

setDefaultPermissions

`public void **setDefaultPermissions**(PermissionInfo[] permissions)`

Sets the default permissions.

These are the permissions granted to any bundle that does not have permissions assigned to its location.

Parameters:

`permissions` – The default permissions.

Throws:

`java.lang.SecurityException` – if the caller does not have the `AdminPermission`.

4.4 org.osgi.service.permissionadmin
Class PermissionInfo

---

public class **PermissionInfo**
extends java.lang.Object

Permission representation used by the `PermissionAdmin` service.

This class encapsulates three pieces of information: a Permission *type* (class name), which must be a subclass of `java.security.Permission`, and the *name* and *actions* arguments passed to its constructor.

In order for a permission represented by a `PermissionInfo` to be instantiated and considered during a permission check, its Permission class must be available from the system classpath or an exported package. This means that the instantiation of a permission represented by a `PermissionInfo` may be delayed until its Permission class has been exported to the framework.

Constructor Summary
PermissionInfo(java.lang.String encodedPermission) Constructs a PermissionInfo from the Permission encoded in the given string, which was obtained by calling `getEncoded()`.
PermissionInfo(java.lang.String type, java.lang.String name, java.lang.String actions) Constructs a PermissionInfo from the given type, name, and actions.

Method Summary	
boolean	**equals**(java.lang.Object obj) Determines the equality of two `PermissionInfo` objects.
java.lang.String	**getActions**() Returns the actions of the permission represented by this `PermissionInfo`.
java.lang.String	**getEncoded**() Returns the string encoding of this `PermissionInfo` in a form suitable for restoring this `PermissionInfo`.

`java.lang.String`	**`getName()`** Returns the name of the permission represented by this `PermissionInfo`.
`java.lang.String`	**`getType()`** Returns the fully qualified class name of the permission represented by this `PermissionInfo`.
`int`	**`hashCode()`** Returns the hash code value for this object.
`java.lang.String`	**`toString()`** Returns the string representation of this `PermissionInfo`.

Constructor Detail

PermissionInfo

```
public PermissionInfo(java.lang.String type,
                      java.lang.String name,
                      java.lang.String actions)
```

Constructs a PermissionInfo from the given type, name, and actions.

Parameters:

`type` − The fully qualified class name of the Permission represented by this PermissionInfo, which must be a subclass of `java.security.Permission` and must define a 2−argument constructor that takes a *name* string and an *actions* string.

`name` − The permission name that will be passed as the first argument to the constructor of the Permission class identified by `type`.

`actions` − The permission actions that will be passed as the second argument to the constructor of the Permission class identified by `type`.

PermissionInfo

```
public PermissionInfo(java.lang.String encodedPermission)
```

Constructs a PermissionInfo from the Permission encoded in the given string, which was obtained by calling `getEncoded()`.

Parameters:

`encodedPermission` − The encoded `PermissionInfo`.

Throws:

`java.lang.IllegalArgumentException` − if encodedPermission is not properly formatted.

See Also:

`getEncoded()`

Method Detail

getEncoded

`public final java.lang.String getEncoded()`

Returns the string encoding of this `PermissionInfo` in a form suitable for restoring this `PermissionInfo`.

The encoding format is:

```
     (type)
```
or
```
     (type "name")
```
or
```
     (type "name" "actions")
```

where *name* and *actions* are strings that are encoded for proper parsing. Specifically, the ", \, carriage return, and linefeed characters are escaped using \", \\, \r, \n respectively.

Returns:
The string encoding of this `PermissionInfo`.

---

toString

`public java.lang.String toString()`

Returns the string representation of this `PermissionInfo`. The string is created by calling the `getEncoded` method on this `PermissionInfo`.

Overrides:
`toString` in class `java.lang.Object`

Returns:
The string representation of this `PermissionInfo`.

---

getType

`public final java.lang.String getType()`

Returns the fully qualified class name of the permission represented by this `PermissionInfo`.

Returns:

The fully qualified class name of the permission represented by this `PermissionInfo`.

---

getName

```
public final java.lang.String getName()
```

Returns the name of the permission represented by this `PermissionInfo`.

Returns:

The name of the permission represented by this `PermissionInfo`, or `null` if the permission does not have a name.

---

getActions

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

Returns the actions of the permission represented by this `PermissionInfo`.

Returns:

The actions of the permission represented by this `PermissionInfo`, or `null` if the permission does not have any actions associated with it.

---

equals

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

Determines the equality of two `PermissionInfo` objects. This method checks that specified object has the same type, name and actions as this `PermissionInfo` object.

Overrides:

`equals` in class `java.lang.Object`

Parameters:

`obj` − The object to test for equality with this `PermissionInfo` object.

Returns:

`true` if *`obj`* is a `PermissionInfo`, and has the same type, name and actions as this `PermissionInfo` object; `false` otherwise.

---

hashCode

```
public int hashCode()
```

Returns the hash code value for this object.

Overrides:

`hashCode` in class `java.lang.Object`

Returns:

A hash code value for this object.
