RFC 120 - Security Enhancements

Final

41 Pages

Abstract

This RFC proposes the ability to deny access to resources. The current system only allows the granting of privileges. Adding the ability to deny privileges leads to simpler administration of security in many use cases and hence a more secure system.

0 Document Information

0.1 Table of Contents

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].

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

0.3 Revision History

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

Revision	Date	Comments
Initial	Aug 8, 2007	Initial revision. Covers only a few enhancements as well as a discussion of a mechanism that has been considered but rejected John Wells, BEA Systems, Inc., jwells@bea.com
0.2	Dec 14, 2007	Clean up document after initial review. Add support for Boolean condition expressions. Include more use case examples.
0.3	Jan. 10, 2008	Yet another change in direction
0.4	Feb. 7, 2008	Remove ABSTAIN and create an atomic update API
0.5	Feb. 15, 2008	Change to a different style of update API
0.6	Mar. 10, 2008	Cleanup to the API based on review comments
0.7	May 7, 2008	More API cleanup
0.8	Jun 25, 2008	Add better algorithm description, fix how postponed conditions should work and fix the example to make more sense
0.9	Jul 23, 2008	Algorithm for postponed conditions
0.91	Aug 05, 2008	Changes to simplify postponed conditions algorithm + updated javadoc Thomas Watson, IBM, tjwatson@us.ibm.com
0.92	Aug 15, 2008	Small fix to postponed algorithm from John Wells Thomas Watson, IBM, tjwatson@us.ibm.com
1.0	12 Dec 2008	No changes. Marked Final for CPEG voting. BJ Hargrave. hargrave@us.ibm.com

1 Introduction

One reasonable property of many “enterprise” servers is that they host user code along with vendor supplied “system” code within the same Java VM. In order to do this there needs to be some reasonable way to allow system code to separate itself from the user code so that the system code can safely offer services, packages and other resources to which it does not wish user code to have access.

This RFC is a response to RFP-78 ([3]). It extends the existing security tuple table, while maintaining backward compatibility with all existing API and security file formats.

2 Application Domain

From [3]:

The application domain is OSGi server applications where third party bundles may be deployed alongside trusted bundles from the application server provider. The ability to split the set of bundles into categories like these is already well defined in the Conditional Permission Admin Service.

2.1 Terminology + Abbreviations

- Application Server: In this paper an application server refers to a running OSGi system that has trusted system bundles written by the application server vendor running alongside third-party user bundles that are provided by the end customer (i.e., not the application server vendor). It does not imply a JEE, .NET or database server, but simply a server whose job it is to host other services and provide them a consistent set of system services upon which they can rely.

- Conditional Permission Service: This is the service defined in Chapter 9 of the OSGi Service Platform Core Specification (R4). It defines conditions, permissions and how security is applied in the R4 OSGi platform.

3 Problem Description

From [3]:

The existing permission classes in OSGi do not have the flexibility needed to handle certain use cases. This includes but is not limited to

- org.osgi.framework.PackagePermission

- org.osgi.framework.ServicePermission

- org.osgi.service.event.TopicPermission

- org.osgi.service.wireadmin.WirePermission
- org.osgi.service.cm.ConfigurationPermission

Providing these capabilities on the OSGi platform will facilitate the adoption of OSGi into enterprise application server environments.

4 Requirements

From [3]:

1. The solution MUST enable restricting certain bundles from offering a service, without having to list all of the services that bundle might offer.
2. The solution MUST enable restricting certain bundles from getting a reference to a service, without having to list all of the other services that bundle might want access to.
3. The solution MUST enable restricting certain bundles from importing or exporting a package, without having to list all of the packages a bundle might import or export.
4. The solution MAY allow for a general Permission denial model
5. The solution MUST NOT alter the bundle programming model
6. The solution MAY boost performance by allowing for Permission implies decisions to be cached
7. Any RFC written MUST support all existing OSGi execution environments

5 Technical Solution

5.1 Ordered Table with Decision Column Discussion

In this solution we change the security table in two ways. Firstly, we make it an ordered table. Secondly, we add another column to the table which determines if the decision of the row is to “allow” access to the resource or “deny” access to the resource.

A few example use cases will make this clear.

In the first use case ACME, a provider of system software, would like to allow all other bundles access to the com.acme package family but restrict them from accessing the “com.acme.secret” and “com.acme.sauce” packages.

Name	Conditions	Permissions	Decision

R1	NOT Signed by ACME	Package(“com.acme.secret”)
		Package(“com.acme.sauce”)
			DENY

R2	Signed by ACME	Package(“com.acme.*”)	ALLOW
R3	<Empty>	Package(“com.acme.*”)	ALLOW

Suppose ACME is attempting to access package “com.acme.secret”. Row R1 will not be evaluated since the condition does not match. Row R2 will be evaluated, and since Package(“com.acme.*”) implies “com.acme.secret” this check will succeed.

Now suppose Iona is attempting to access package “com.acme.secret”. Row R1 would be evaluated and since Package(“com.acme.secret”) implies “com.acme.secret” this check would return DENY and would fail as expected.

If instead Iona is attempting to access package “com.acme.service” then Row R1 would be evaluated. However, since neither Package(“com.acme.secret”) nor Package(“com.acme.sauce”) implies “com.acme.service” the row will not be evaluated (in effect the decision of this row is to ABSTAIN). Row R2 will be skipped (since the condition does not match) and row R3 will be evaluated. Since Package(“com.acme.*”) implies “com.acme.service” the decision of ALLOW will be taken and the permission check will succeed.

Note that the next table is equivalent to the table above:

Name	Conditions	Permissions	Decision

R1	NOT Signed by ACME	Package(“com.acme.secret”)
		Package(“com.acme.sauce”)
			DENY

R2	<Empty>	Package(“com.acme.*”)	ALLOW

One use case (which is not in the original set of use cases) that is handled cleanly by this solution is the friends use case. In this use case we want to express the following scenario:

1. Pepsi wants to deny permissions to com.pepsi.* for everyone but Pepsi
2. Pepsi wants to allow Coke to have permission to com.pepsi.friends.*
3. All people should have access to all other packages

The following table enables the above use case:

All Page Within This Box

Name	Conditions	Permissions	Decision
R1	Signed by Coke	Package(“com.pepsi.friends.*”)	ALLOW
R2	Not Signed by Pepsi	Package(“com.pepsi.*”)	DENY
R3	<Empty>	Package(“*”)	ALLOW

The reason that the above use case works is that the table is now ordered, and will be evaluated in row order.

If Coke attempts to access the “com.pepsi.friends.foo” package then row R1 will be evaluated first and since Package(“com.pepsi.friends.*”) implies package “com.pepsi.friends.foo” the ALLOW decision is taken and the check will succeed.

If Coke attempts to access the “com.pepsi.secret” package then row R1 will be evaluated first and since Package(“com.pepsi.friends.*”) does not imply package “com.pepsi.secret” the row is not evaluated (in essence row R1 abstains from the decision). Row R2 also applies to Coke and since Package(“com.pepsi.*”) implies package “com.pepsi.secret” the decision of DENY is found and hence this check would fail (properly).

If I am Pepsi on the other hand, if I attempt to access package “com.pepsi.friends” or package “com.pepsi.secret” then neither rows R1 or R2 apply, and only row R3 will be considered. Since Package(“*”) implies both the packages listed above the decision of ALLOW is found and hence this check would succeed.

Also, RC Cola, who is *not* a friend of Pepsi, does not have access to either “com.pepsi.friends.*” or “com.pepsi.*” but does have access to all the other packages in the system.

5.2 Conditional Permission Table Requirements

A fourth column shall be added to the conceptual Conditional Permission table. The cells in this column shall take a single java.lang.String object. The allowable values for this string shall be “allow” or “deny”. The allowable decision strings shall be added to the org.osgi.service.condpermadmin.ConditionalPermissionInfoBase class.

Previously the order of the rows in the conceptual Conditional Permission table were not significant. The rows of the conceptual Conditional Permission table shall now be ordered by the index of the row. In other words, all rows shall be evaluated in index order (lowest to highest). This includes rows with postponed conditions.

A permission check starts when the Security Manager checkPermission method is called with permission P as argument. This Security Manager must be implemented by the Framework and is therefore called the Framework Security Manager; it must be fully integrated with the Conditional Permission Admin service.

The Framework Security Manager must get the Access Control Context in effect. It must call the AccessController getContext() method to get the default context if it is not passed a specific context.

The AccessControlContext checkPermission method must then be called, which causes the call stack to be traversed. At each stack level the Bundle Protection Domain of the calling class is evaluated for the permission P using the ProtectionDomain implies method. This complete evaluation must take place on the same thread.

P must be implied by the local permissions of the Bundle Protection Domain. If this is not the case, the check must end with a failure.

The Bundle Protection Domain must now decide which rows in its instantiated conditional permission table are applicable and imply P.

It must therefore execute the following instructions or reach the same result in an alternative way:

- For each row R in the domain's instantiated conditional permission table:
  - If R has immediate conditions, evaluate all these immediate Condition objects. If any of these objects is not satisfied, continue with next row.
  - If R's permissions do not imply P, continue with the next row.
  - If R contains postponed Condition objects then add R to the end of the postpone list for the domain's instantiated conditional permission table (which will postpone the evaluation of R) and continue with the next row.
  - Otherwise:
    - If no rows have been postponed for the domain then return true if R's decision is Grant; otherwise return false if R's decision is Deny.
    - Otherwise the postponed list for the domain's instantiated conditional permission table shall be inspected from the END to the START removing all postponed rows that have the same decision as R's decision until a postponed row is encountered with the opposite decision
    - If all postponed rows are removed then return true if R's decision is Grant; otherwise return false if R's decision is Deny.
    - Otherwise the remaining postponed rows along with R's immediate decision must be postponed and true must be returned
- After all rows have been processed and no immediate decision has been found
  - If there are no postponements then return false
  - Otherwise the postponed list for the domain's instantiated conditional permission table shall be inspected from END to the START removing all postponed rows that have a DENY decision until a postponed row is found with an ALLOW decision. If all postponed rows are removed (i.e. no postponed rows with an ALLOW decision exist) then return false; otherwise return true.

After the Framework Security Manager has called the checkPermission method of the Access Control Context, it must decide to fail or handle the postponed rows for each domain. If this method returns false, then the Framework Security Manager's checkPermission method fails.

If it returns true, there could still be a list of postponed rows for each Bundle Protection Domain's instantiated conditional permission table. Each of these postponed rows already imply permission P, otherwise they must not have been placed on the postponed list for the domain's instantiated conditional permission table. However, their Condition objects still need to be satisfied before the decision of the row can apply to the security check.

Frameworks are free to implement an algorithm that finds optimal ways to permute the postponed rows from the different domains involved in the permission check. The end result must return the same answer as the following algorithm for processing postponed conditions.

- For each domain D that has an ordered list of postponed rows
  - For each postponed row R
    - For each postponed condition C
      - Call Condition.isSatisified(Condition[], Dictionary) with the single condition instance C. Each call to Condition.isSatisified(Condition[], Dictionary) for a single condition type must use the same Dictionary instance during a single permission check.

All Page Within This Box

            - If Condition.isSatisified(Condition[], Dictionary) returns false then continue to the next postponed row; Otherwise continue to the next condition.
        - If the decision is Grant then continue to the next domain
        - If the decision is Deny then throw a SecurityException
    - If there was an immediate decision recorded along with the postponed rows of domain D then
        - If the immediate decision is Grant continue to the next domain
        - If the decision is Deny then throw a SecurityException
    - Otherwise no rows apply to the permission check for this domain; a SecurityException must be thrown.
- If every domain is processed without throwing a SecurityException then the permission is granted.

The algorithm described above is descriptive and other implementations of how the postponed lists work may be used, as long as the end goal of having the security table behave as if it is ordered (even if the conditions are not actually evaluated in the given order) is achieved.

5.3 Negative Condition Requirements

The org.osgi.service.condpermadmin.BundleLocationCondition and org.osgi.service.condpermadmin.BundleSignerCondition shall have an optional second string added to their initialization arguments. If this string is equal to “!” then the Condition shall return the logical NOT of the result of the match of the first string (the DN in the case of the BundleSignerCondition and the bundle location in the case of the BundleLocationCondition).

5.4 Atomic API Discussion

In order to support an ordered permission table it is necessary to support an atomic style of updating that table. In particular multiple entries may need to be added or removed from the table in a single operation in order to avoid either granting or denying too many rights. However, the original API was not designed for atomic update. In particular, the “delete” method on the ConditionalPermissionInfo objects as the means to remove items from the table does not lend itself to atomicity.

In order to support atomic updates to the table a new object called the ConditionalPermissionAdminUpdate (or just “Update” for short) has been created. The Update is originally created with a copy of the existing Permission Table. There may be any number of Update objects in the system based on the running Permission Table. However an Update will only successfully commit if the running permission table has not changed.

The ConditionalPermissionInfo interface has been split into two interfaces, the ConditionalPermissionInfoBase interface and the existing ConditionalPermissionInfo interface. All of the methods previously in the ConditionalPermissionInfo interface have been moved to the ConditionalPermissionInfoBase interface with the exception of the delete method, which has remained in the ConditionalPermissionInfo interface. The ConditionalPermissionInfo interface extends the ConditionalPermissionInfoBase interface. In this way both source and binary compatibility is achieved, while still allowing an atomic coding paradigm. (Note that source compatibility does **not** extend to the use of reflection, which may now get different results).

The Update’s List can be modified by the code without affecting the currently running permission table. At the time the commit method is called on the Update’s list will become the new permission table as long as no other

All Page Within This Box

update has been committed on the Permission Table since the Update was created. There is no requirement that the commit method be called on an Update object.

While the Update mechanism helps satisfy the requirement for atomic updates to the permission table, there are some idiosyncrasies associated with the approach. For example, since there is no lock it is not possible to write code using this API that can be completely isolated from the possibility of losing the Update race. Therefore to be completely correct all code using this new API must handle the case where the Update race is lost. This may or may not be difficult to achieve in general code.

The exact mechanism and behavior for the atomic updates of the permission table can be found in the javadoc section of this document.

5.5 Javadoc

Several classes need to be modified to support the specified requirements. The javadoc should be considered a binding part of this specification. Note that only API that has been modified or added is included in this document. All API not in this document shall continue to behave as per prior art.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

5.5.1 org.osgi.service.condpermadmin
Class BundleLocationCondition

```
java.lang.Object
    org.osgi.service.condpermadmin.BundleLocationCondition
```

```
public class BundleLocationCondition
extends java.lang.Object
```

Condition to test if the location of a bundle matches or does not match a pattern. Since the bundle's location cannot be changed, this condition is immutable.

Pattern matching is done according to the filter string matching rules.

Version:

$Revision: 5185 $

Method Summary

`static Condition`	`getCondition(Bundle bundle, ConditionInfo info)` Constructs a condition that tries to match the passed Bundle's location to the location pattern.

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait,`

```
wait, wait
```

Method Detail

5.5.1.1 getCondition

```
public static Condition getCondition(Bundle bundle,
                                     ConditionInfo info)
```

Constructs a condition that tries to match the passed Bundle's location to the location pattern.

Parameters:

`bundle` - The Bundle being evaluated.

`info` - The ConditionInfo from which to construct the condition. The ConditionInfo must specify one or two arguments. The first argument of the ConditionInfo specifies the location pattern against which to match the bundle location. Matching is done according to the filter string matching rules. Any '*' characters in the first argument are used as wildcards when matching bundle locations unless they are escaped with a '\' character. The Condition is satisfied if the bundle location matches the pattern. The second argument of the ConditionInfo is optional. If a second argument is present and equal to "!", then the satisfaction of the Condition is negated. That is, the Condition is satisfied if the bundle location does NOT match the pattern. If the second argument is present but does not equal "!", then the second argument is ignored.

Returns:

Condition object for the requested condition.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

5.5.2 org.osgi.service.condpermadmin
Class BundleSignerCondition

```
java.lang.Object
    org.osgi.service.condpermadmin.BundleSignerCondition
```

```
public class BundleSignerCondition
extends java.lang.Object
```

Condition to test if the signer of a bundle matches or does not match a pattern. Since the bundle's signer can only change when the bundle is updated, this condition is immutable.

The condition expressed using a single String that specifies a Distinguished Name (DN) chain to match bundle signers against. DN's are encoded using IETF RFC 2253. Usually signers use certificates that are issued by certificate authorities, which also have a corresponding DN and certificate. The certificate authorities can form a chain of trust where the last DN and certificate is known by the framework. The signer of a bundle is expressed as signers DN followed by the DN of its issuer followed by the DN of the next issuer until the DN of the root certificate authority. Each DN is separated by a semicolon.

A bundle can satisfy this condition if one of its signers has a DN chain that matches the DN chain used to construct this condition. Wildcards (`*') can be used to allow greater flexibility in specifying the DN chains. Wildcards can be used in place of DNs, RDNs, or the value in an RDN. If a wildcard is used for a value of an RDN, the value must be exactly "*" and will match any value for the corresponding type in that RDN. If a wildcard is used for a RDN, it must be the first RDN and will match any number of RDNs (including zero RDNs).

Version:

$Revision: 5185 $

Method Summary

`static Condition`	`getCondition(Bundle bundle, ConditionInfo info)` Constructs a Condition that tries to match the passed Bundle's location to the location pattern.

Methods inherited from class java.lang.Object
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Method Detail

5.5.2.1 getCondition

```
public static Condition getCondition(Bundle bundle,
                                     ConditionInfo info)
```

Constructs a Condition that tries to match the passed Bundle's location to the location pattern.

Parameters:

`bundle` - The Bundle being evaluated.

`info` - The ConditionInfo from which to construct the condition. The ConditionInfo must specify one or two arguments. The first argument of the ConditionInfo specifies the chain of distinguished names pattern to match against the signer of the bundle. The Condition is satisfied if the signer of the bundle matches the pattern. The second argument of the ConditionInfo is optional. If a second argument is present and equal to "!", then the satisfaction of the Condition is negated. That is, the Condition is satisfied if the signer of the bundle does NOT match the pattern. If the second argument is present but does not equal "!", then the second argument is ignored.

Returns:

A Condition which checks the signers of the specified bundle.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

5.5.3 org.osgi.service.condpermadmin Interface Condition

```
public interface Condition
```

The interface implemented by a Condition. Conditions are bound to Permissions using Conditional Permission Info. The Permissions of a ConditionalPermission Info can only be used if the associated Conditions are satisfied.

Version:

$Revision: 5184 $

Field Summary

`static Condition`	**`FALSE`** A Condition object that will always evaluate to false and that is never postponed.
`static Condition`	**`TRUE`** A Condition object that will always evaluate to true and that is never postponed.

Method Summary

`boolean`	**`isMutable`**`()` Returns whether the Condition is mutable.
`boolean`	**`isPostponed`**`()` Returns whether the evaluation must be postponed until the end of the permission check.
`boolean`	**`isSatisfied`**`()` Returns whether the Condition is satisfied.
`boolean`	**`isSatisfied`**`(Condition[] conditions, java.util.Dictionary context)` Returns whether a the set of Conditions are satisfied.

Field Detail

5.5.3.1 TRUE

```
static final Condition TRUE
```

A Condition object that will always evaluate to true and that is never postponed.

5.5.3.2 FALSE

```
static final Condition FALSE
```

A Condition object that will always evaluate to false and that is never postponed.

Method Detail

5.5.3.3 isPostponed

```
boolean isPostponed()
```

Returns whether the evaluation must be postponed until the end of the permission check. This method returns `true` if the evaluation of the Condition must be postponed until the end of the permission check. If this method returns `false`, this Condition must be able to directly answer the `isSatisfied()` method. In other words, isSatisfied() will return very quickly since no external sources, such as for example users, need to be consulted.

Returns:

`true` to indicate the evaluation must be postponed. Otherwise, `false` if the evaluation can be immediately performed.

5.5.3.4 isSatisfied

```
boolean isSatisfied()
```

Returns whether the Condition is satisfied.

Returns:

`true` to indicate the Conditions is satisfied. Otherwise, `false` if the Condition is not satisfied.

5.5.3.5 isMutable

```
boolean isMutable()
```

Returns whether the Condition is mutable.

Returns:

`true` to indicate the value returned by `isSatisfied()` can change. Otherwise, `false` if the value returned by `isSatisfied()` will not change.

5.5.3.6 isSatisfied

```
boolean isSatisfied(Condition[] conditions,
                    java.util.Dictionary context)
```

Returns whether a the set of Conditions are satisfied. Although this method is not static, it must be implemented as if it were static. All of the passed Conditions will be of the same type and will correspond to the class type of the object on which this method is invoked.

Parameters:

`conditions` - The array of Conditions.
`context` - A Dictionary object that implementors can use to track state. If this method is invoked multiple times in the same permission evaluation, the same Dictionary will be passed multiple times. The SecurityManager treats this Dictionary as an opaque object and simply creates an empty dictionary and passes it to subsequent invocations if multiple invocations are needed.
Returns:
`true` if all the Conditions are satisfied. Otherwise, `false` if one of the Conditions is not satisfied.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

5.5.4 org.osgi.service.condpermadmin
Interface ConditionalPermissionAdmin

```
public interface ConditionalPermissionAdmin
```

Framework service to administer Conditional Permissions. Conditional Permissions can be added to, retrieved from, and removed from the framework. Conditional Permissions are conceptually managed in an ordered table called the Conditional Permission Table.

Version:
$Revision: 5188 $

Method Summary

`ConditionalPermissionInfo`	**`addConditionalPermissionInfo`**`(ConditionInfo[] conds, PermissionInfo[] perms)` **Deprecated.** *Since 1.1. Use `ConditionalPermissionsUpd` instead.*
`ConditionalPermissionInfoBase`	**`createConditionalPermissionInfoBase`**`(java.lang.String na ConditionInfo[] conditions, PermissionInfo[] permissio java.lang.String decision)` Creates a ConditionalPermissionInfoBase with the specified fields.
`ConditionalPermissionsUpdate`	**`createConditionalPermissionsUpdate`**`()` Creates an update for the Conditional Permission Table.
`Java.security.AccessControlContext`	**`getAccessControlContext`**`(java.lang.String[] signers)` Returns the Access Control Context that corresponds to the spec signers.
`ConditionalPermissionInfo`	**`getConditionalPermissionInfo`**`(java.lang.String name)`

	Return the Conditional Permission Info with the specified name.
java.util.Enumeration	**getConditionalPermissionInfos**() Returns the Conditional Permission Infos from the Conditi Permission
ConditionalPermissionInfo	**setConditionalPermissionInfo**(java.lang.String name, ConditionInfo[] conds, PermissionInfo[] per **Deprecated.** *Since 1.1. Use ConditionalPermissionsUpd instead.*

Method Detail

5.5.4.1 addConditionalPermissionInfo

```
ConditionalPermissionInfo addConditionalPermissionInfo(ConditionInfo[] conds,
                                                       PermissionInfo[] perms)
```

Deprecated. *Since 1.1. Use ConditionalPermissionsUpdate instead.*
Create a new Conditional Permission Info in the Conditional Permission Table.

The Conditional Permission Info will be given a unique, never reused name. This entry will be added at the beginning of the Conditional Permission Table with a grant decision of ALLOW.

Since this method changes the Conditional Permission Table any ConditionalPermissionsUpdates that were created prior to calling this method can no longer be committed.

Parameters:
conds - The Conditions that need to be satisfied to enable the corresponding Permissions.
perms - The Permissions that are enabled when the corresponding Conditions are satisfied.
Returns:
The ConditionalPermissionInfo for the specified Conditions and Permissions.
Throws:
java.lang.SecurityException - If the caller does not have AllPermission.

5.5.4.2 setConditionalPermissionInfo

```
ConditionalPermissionInfo setConditionalPermissionInfo(java.lang.String name,
                                                       ConditionInfo[] conds,
                                                       PermissionInfo[] perms)
```

Deprecated. *Since 1.1. Use ConditionalPermissionsUpdate instead.*
Set or create a Conditional Permission Info with a specified name in the Conditional Permission Table.

If the specified name is null, a new Conditional Permission Info must be created and will be given a unique, never reused name. If there is currently no Conditional Permission Info with the specified name, a new Conditional Permission Info must be created with the specified name. Otherwise, the Conditional Permission Info with the specified name must be updated with the specified Conditions and Permissions. If a new entry was created in the Conditional Permission Table it will be added at the beginning of the table with a grant decision of ALLOW.

Since this method changes the underlying permission table any `ConditionalPermissionsUpdate`s that were created prior to calling this method can no longer be committed.

Parameters:
`name` - The name of the Conditional Permission Info, or `null`.
`conds` - The Conditions that need to be satisfied to enable the corresponding Permissions.
`perms` - The Permissions that are enabled when the corresponding Conditions are satisfied.
Returns:
The ConditionalPermissionInfo that for the specified name, Conditions and Permissions.
Throws:
`java.lang.SecurityException` - If the caller does not have `AllPermission`.

5.5.4.3 getConditionalPermissionInfos

`java.util.Enumeration` **`getConditionalPermissionInfos`**`()`

Returns the Conditional Permission Infos from the Conditional Permission

The returned Enumeration will return elements in the order they are kept in the Conditional Permission Table.

The Enumeration returned is based on a copy of the Conditional Permission Table and therefore will not throw exceptions if the Conditional Permission Table is changed during the course of reading elements from the Enumeration.

Returns:
An enumeration of the Conditional Permission Infos that are currently in the Conditional Permission Table.

5.5.4.4 getConditionalPermissionInfo

`ConditionalPermissionInfo` **`getConditionalPermissionInfo`**`(java.lang.String name)`

Return the Conditional Permission Info with the specified name.
Parameters:
`name` - The name of the Conditional Permission Info to be returned.
Returns:
The Conditional Permission Info with the specified name.

5.5.4.5 getAccessControlContext

`java.security.AccessControlContext`
`getAccessControlContext``(java.lang.String[] signers)`

Returns the Access Control Context that corresponds to the specified signers.
Parameters:
`signers` - The signers for which to return an Access Control Context.
Returns:
An `AccessControlContext` that has the Permissions associated with the signer.

5.5.4.6 *createConditionalPermissionsUpdate*

```
ConditionalPermissionsUpdate createConditionalPermissionsUpdate()
```

Creates an update for the Conditional Permission Table. The update is a working copy of the current Conditional Permission Table. If the running Conditional Permission Table is modified before commit is called on the returned update, then the call to commit will fail. That is, the commit method will return false and no change will be made to the running Conditional Permission Table. There is no requirement that commit is eventually called on the returned update.

Returns:

An update for the Conditional Permission Table.

Since:

1.1

5.5.4.7 *createConditionalPermissionInfoBase*

```
ConditionalPermissionInfoBase
createConditionalPermissionInfoBase(java.lang.String name,

ConditionInfo[] conditions,

PermissionInfo[] permissions,

java.lang.String decision)
```

Creates a ConditionalPermissionInfoBase with the specified fields.

Parameters:

`name` - The name of the created ConditionalPermissionInfoBase or `null` to have a unique name generated when the created ConditionalPermissionInfoBase is committed in an update to the Conditional Permission Table.

`conditions` - The Conditions that need to be satisfied to enable the corresponding Permissions.

`permissions` - The Permissions that are enabled when the corresponding Conditions are satisfied.

`decision` - One of the following values:

- `allow`
- `deny`

Returns:

A ConditionalPermissionInfoBase object suitable for insertion in a `ConditionalPermissionsUpdate`.

Throws:

`java.lang.IllegalArgumentException` - If the decision string is invalid.

Since:

1.1

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS

FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS

SUMMARY: NESTED | FIELD | CONSTR | METHOD

FRAMES NO FRAMES All Classes

DETAIL: FIELD | CONSTR | METHOD

5.5.5 org.osgi.service.condpermadmin Interface ConditionalPermissionInfo

All Superinterfaces:

ConditionalPermissionInfoBase

```
public interface ConditionalPermissionInfo
extends ConditionalPermissionInfoBase
```

A binding of a set of Conditions to a set of Permissions. Instances of this interface are obtained from the Conditional Permission Admin service.

Version:

$Revision: 5188 $

Field Summary

Fields inherited from interface org.osgi.service.condpermadmin.ConditionalPermissionInfoBase
`ALLOW`, `DENY`

Method Summary

`void`	**`delete()`** Removes this Conditional Permission Info from the Conditional Permission Table.

Methods inherited from interface org.osgi.service.condpermadmin.ConditionalPermissionInfoBase
`getConditionInfos`, `getGrantDecision`, `getName`, `getPermissionInfos`

Method Detail

5.5.5.1 delete

```
void delete()
```

Removes this Conditional Permission Info from the Conditional Permission Table.

Since this method changes the underlying permission table any `ConditionalPermissionsUpdate`s that were created prior to calling this method can no longer be committed.

Throws:

`java.lang.SecurityException` - If the caller does not have `AllPermission`.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

5.5.6 org.osgi.service.condpermadmin

Interface ConditionalPermissionInfoBase

All Known Subinterfaces:

ConditionalPermissionInfo

```
public interface ConditionalPermissionInfoBase
```

A binding of a set of Conditions to a set of Permissions. Instances of this interface are obtained from the Conditional Permission Admin service.

Since:

1.1

Version:

$Revision: 5185 $

Field Summary

`static java.lang.String`	**`ALLOW`** This string is used to indicate that a row in the conditional permission admin table should return a grant decision of ALLOW if the conditions are all satisfied and at least one of the permissions is implied.
`static java.lang.String`	**`DENY`** This string is used to indicate that a row in the conditional permission admin table should return a grant decision of DENY if the conditions are all satisfied and at least one of the permissions is implied.

Method Summary

`ConditionInfo[]`	**`getConditionInfos`**`()` Returns the Condition Infos for the Conditions that must be satisfied to enable the Permissions.
`java.lang.String`	**`getGrantDecision`**`()` Returns the grant decision for this Conditional Permission Info.

`java.lang.String`	`getName()` Returns the name of this Conditional Permission Info.
`PermissionInfo[]`	`getPermissionInfos()` Returns the Permission Infos for the Permission in this Conditional Permission Info.

Field Detail

5.5.6.1 ALLOW

`static final java.lang.String` **`ALLOW`**

This string is used to indicate that a row in the conditional permission admin table should return a grant decision of ALLOW if the conditions are all satisfied and at least one of the permissions is implied.

See Also:

Constant Field Values

5.5.6.2 DENY

`static final java.lang.String` **`DENY`**

This string is used to indicate that a row in the conditional permission admin table should return a grant decision of DENY if the conditions are all satisfied and at least one of the permissions is implied.

See Also:

Constant Field Values

Method Detail

5.5.6.3 getConditionInfos

`ConditionInfo[]` **`getConditionInfos`**`()`

Returns the Condition Infos for the Conditions that must be satisfied to enable the Permissions.

Returns:

The Condition Infos for the Conditions in this Conditional Permission Info.

5.5.6.4 getPermissionInfos

`PermissionInfo[]` **`getPermissionInfos`**`()`

Returns the Permission Infos for the Permission in this Conditional Permission Info.

Returns:

The Permission Infos for the Permission in this Conditional Permission Info.

5.5.6.5 getGrantDecision

`java.lang.String` **`getGrantDecision`**`()`

Returns the grant decision for this Conditional Permission Info.

Returns:

One of the following values:

- `allow` - The grant decision is allow.
- `deny` - The grant decision is DENY.

5.5.6.6 getName

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

Returns the name of this Conditional Permission Info.

Returns:

The name of this Conditional Permission Info.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS **NEXT CLASS** **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

5.5.7 org.osgi.service.condpermadmin Interface ConditionalPermissionsUpdate

```
public interface ConditionalPermissionsUpdate
```

Update the Conditional Permission Table. There may be many update objects in the system at one time. If commit is called and the Conditional Permission Table has been modified since this update was created, then the call to commit will fail and this object should be discarded.

Since:

1.1

Version:

$Revision: 5188 $

Method Summary

`boolean`	`commit()` Commit the update.
`Java.util.List`	`getConditionalPermissionInfoBases()` This method returns the list of `ConditionalPermissionInfoBase`s for this update.

Method Detail

5.5.7.1 getConditionalPermissionInfoBases

`java.util.List` **`getConditionalPermissionInfoBases`**`()`

This method returns the list of `ConditionalPermissionInfoBase`s for this update. This list is originally based on the Conditional Permission Table at the time this update was created. The list returned by this method will be replace the Conditional Permission Table if commit is called and is successful.

The elements of the list must NOT be instances of type `ConditionalPermissionInfo`, but must rather be of type `ConditionalPermissionInfoBase`. This is to ensure the `delete` method cannot be mistakenly used.

The list returned by this method is ordered and the most significant table entry is the first entry in the list.

Returns:
A `List` of the Conditional Permission Info Bases which represent the Conditional Permissions maintained by this update. Modifications to this list will not affect the Conditional Permission Table until successfully committed. The elements in this list must be of type `ConditionalPermissionInfoBase`. The list may be empty if the Conditional Permission Table was empty when this update was created.

5.5.7.2 commit

`boolean` **`commit`**`()`

Commit the update. If no changes have been made to the Conditional Permission Table since this update was created, then this method will replace the Conditional Permission Table with this update's Conditional Permissions. This method may only be successfully called once on this object.

If any of the `ConditionalPermissionInfoBase` objects in the update list has `null` as a name it will be replaced with a `ConditionalPermissionInfoBase` object that has a generated name which is unique within the list.

No two entries in this update's Conditional Permissions may have the same name. Other consistency checks may also be performed. If the update's Conditional Permissions are determined to be inconsistent in some way then an `IllegalStateException` will be thrown.

This method returns `false` if the Conditional Permission Table has been modified since the creation of this update.

Returns:
`true` if the commit was successful. `false` if the Conditional Permission Table has been modified since the creation of this update.

Throws:
`java.lang.SecurityException` - If the caller does not have `AllPermission`.
`java.lang.IllegalStateException` - If the update's Conditional Permissions are not valid or inconsistent. For example, if this update has two Conditional Permissions in it with the same name, then this exception will be thrown.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

5.5.8 org.osgi.service.condpermadmin Class ConditionInfo

```
java.lang.Object
    org.osgi.service.condpermadmin.ConditionInfo
```

```
public class ConditionInfo
extends java.lang.Object
```

Condition representation used by the Conditional Permission Admin service.

This class encapsulates two pieces of information: a Condition *type* (class name), which must implement `Condition`, and the arguments passed to its constructor.

In order for a Condition represented by a `ConditionInfo` to be instantiated and considered during a permission check, its Condition class must be available from the system classpath.

The Condition class must either:

- Declare a public static `getCondition` method that takes a `Bundle` object and a `ConditionInfo` object as arguments. That method must return an object that implements the `Condition` interface.
- Implement the `Condition` interface and define a public constructor that takes a `Bundle` object and a `ConditionInfo` object as arguments.

Version:

$Revision: 5183 $

Constructor Summary

ConditionInfo(java.lang.String encodedCondition) Constructs a `ConditionInfo` object from the specified encoded `ConditionInfo` string.
ConditionInfo(java.lang.String type, java.lang.String[] args) Constructs a `ConditionInfo` from the specified type and args.

Method Summary

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

	Determines the equality of two `ConditionInfo` objects.
`Java.lang.String[]`	**`getArgs`**`()` Returns arguments of this `ConditionInfo`.
`java.lang.String`	**`getEncoded`**`()` Returns the string encoding of this `ConditionInfo` in a form suitable for restoring this `ConditionInfo`.
`java.lang.String`	**`getType`**`()` Returns the fully qualified class name of the condition represented by this `ConditionInfo`.
`int`	**`hashCode`**`()` Returns the hash code value for this object.
`java.lang.String`	**`toString`**`()` Returns the string representation of this `ConditionInfo`.

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

Constructor Detail

5.5.8.1 ConditionInfo

```
public ConditionInfo(java.lang.String type,
                     java.lang.String[] args)
```

Constructs a `ConditionInfo` from the specified type and args.

Parameters:

`type` - The fully qualified class name of the Condition represented by this `ConditionInfo`.

`args` - The arguments for the Condition. These arguments are available to the newly created Condition by calling the `getArgs()` method.

Throws:

`java.lang.NullPointerException` - If `type` is `null`.

5.5.8.2 ConditionInfo

```
public ConditionInfo(java.lang.String encodedCondition)
```

Constructs a `ConditionInfo` object from the specified encoded `ConditionInfo` string. White space in the encoded `ConditionInfo` string is ignored.

Parameters:

`encodedCondition` - The encoded `ConditionInfo`.

Throws:

`java.lang.IllegalArgumentException` - If the `encodedCondition` is not properly formatted.

See Also:

`getEncoded()`

Method Detail

5.5.8.3 getEncoded

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

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

The encoding format is:

```
[type "arg0" "arg1" ...]
```

where *argN* are strings that are encoded for proper parsing. Specifically, the `"`, `\`, carriage return, and line feed characters are escaped using `\"`, `\\`, `\r`, and `\n`, respectively.

The encoded string contains no leading or trailing whitespace characters. A single space character is used between type and "*arg0*" and between the arguments.

Returns:
The string encoding of this `ConditionInfo`.

5.5.8.4 toString

`public java.lang.String toString()`

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

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

Returns:
The string representation of this `ConditionInfo`.

5.5.8.5 getType

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

Returns the fully qualified class name of the condition represented by this `ConditionInfo`.

Returns:
The fully qualified class name of the condition represented by this `ConditionInfo`.

5.5.8.6 getArgs

`public final java.lang.String[] getArgs()`

Returns arguments of this `ConditionInfo`.

Returns:
The arguments of this `ConditionInfo`. An empty array is returned if the `ConditionInfo` has no arguments.

5.5.8.7 equals

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

Determines the equality of two `ConditionInfo` objects. This method checks that specified object has the same type and args as this `ConditionInfo` object.

Overrides:

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

Parameters:

`obj` - The object to test for equality with this `ConditionInfo` object.

Returns:

`true` if `obj` is a `ConditionInfo`, and has the same type and args as this `ConditionInfo` object; `false` otherwise.

5.5.8.8 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.

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

5.6 Open Issues

5.7 Closed Issues

5.7.1 Default Decision

5.7.1.1 Original Issue

In a system where denying permissions is explicitly and appropriately handled, it is useful to be able to control the decision if no-one has an explicit decision. Currently if no row in the table matches, then the decision taken is to disallow the requested permission. This behavior is fine in a system where rights cannot be denied as well as granted. Since this system allows for permissions to be denied it becomes useful (and would lead to fewer table entries) if the decision taken when no-one has made an explicit decision could be set to allow the request.

One common use-case where this is important is when someone wishes to have the last row in the table turn on all of the permissions. The below table is an example:

Name	Conditions	Permissions	Decision

Name	Conditions	Permissions	Decision
R1	<whatever>	<whatever>	DENY
LastRow	<empty>	java.lang.AllPermission	ALLOW

In this use case, the last row in the table allows all permissions to everyone. This makes the security of the system based on what is denied as opposed to what is granted. However, many of the API (in particular the legacy API) have specified that they add elements to the end of the table. Also, it is very easy to add items to the end of a list, whereas adding things next-to-the end is somewhat trickier. Therefore, it would make the programming model easier if instead there were a “default” decision that was taken if no rows of the table matched. If people knew that the default decision to be taken was ALLOW as opposed to abstain, they could safely add their entries to the end of the list, rather than having to add a row that must always remain at the end of the list.

This enhancement has not been specified pending further discussion of the merits of the proposal.

5.7.1.2 Resolution

Despite the difficulties introduced to the programmer when using an ordered table this enhancement will not be done. The javadoc for the original API that added items to the table have been modified to specify that they will add their entries to the beginning of the table rather than the end so that it might be easier to use the old API in conjunction with the Update API.

5.7.2 An Alternate Update model

5.7.2.1 Original Issue

Another model for atomic updates is one in which the “createUpdate” method can create multiple updates and that only the “commit” method need be atomic. In this mode, each update gets a working copy of the Permission Admin Table and when commit happens successfully its copy becomes the new Permission Admin Table. If multiple Updates are based off of the same conceptual table generation number then only one of the updates would succeed at commit time. All of the other updates would fail at commit time, with an explicit exception being thrown indicating that the update was based on an older generation number. It would be up to the code at that point to re-do the work it had done.

This option is viable and has the advantage that blocking semantics and timeouts need not be specified. It however suffers the drawback that the error case will happen even in properly written code and thus will need to be dealt with by the client code. In both proposals the client code needs to deal with the problems of inconsistent lists and other semantic issues, but in this proposal the code would also have to deal with the hirsute problem of a lost race.

Since the existing proposal seems to be easier to use from a client perspective it has been chosen, pending further discussion of the merits of the multiple-update proposal.

5.7.2.2 Resolution

This model has been adopted in this version of the specification. Version 0.4 of this document proposed a mechanism whereby an exclusive lock could be held, ensuring multiple simultaneous updates could not occur. In this solution there is no supplied methodology to ensure races do not occur. Lost races are reported to the user, who is responsible for redoing the work lost.

5.7.3 Utility methods

5.7.3.1 Original Issue

The methods createInfoBase() and generateUniqueName() are currently specified on the org.osgi.service.condpermadmin.ConditionalPermissionAdminUpdate object. However, these methods have nothing to do with the particular update and would be better served as static methods on some Utility class. There does not seem to be any generic "Utility" class in the OSGi framework. It would be useful to have such a class for methods like these to be specified.

Since this RFC does not appear to be the proper place to propose such a Utility class the methods will remain where they are, pending further discussion.

5.7.3.2 Resolution

Since Java has a deficiency in supporting static style methods, these methods will remain on an interface. However, they have been moved from the Update interface to the Admin interface.

6 Considered Alternatives

The idea here is to have a "DeniablePermission" class that extends BasicPermission. All of the classes listed above that currently extend BasicPermission would be changed to extend DeniablePermission.

This solution has the advantage of being very easy to implement. However, there are subtleties of the syntax which are not desirable. For example, if someone were to specify "*-a.b" and also "*-a.c" resolves to "*". Instead, the user who wanted to restrict both a.b and a.c should have said "*-a.b,a.c". This might confuse people and make it difficult to read and understand the security constraints placed on the Permission.

The javadoc for the DeniablePermission can be found below:

6.1 com.bea.sandbox.security.permission

Class DeniablePermission

```
java.lang.Object
  └java.security.Permission
      └java.security.BasicPermission
          └com.bea.sandbox.security.permission.DeniablePermission
```

All Implemented Interfaces:

Serializable, Guard

```
public class DeniablePermission
extends BasicPermission
```

This is an extension of BasicPermission which understands denials as well as grants.

The name for a DeniablePermission has two parts, the part before the '-' character, and the part after the '-' character. If there is no '-' character in the name then DeniablePermission acts exactly like a BasicPermission. The firt part of the name is the Permission granted, while the second part of the name are the things denied.

The rules for the first part of the name (things granted) follow the same rules as BasicPermission. The first part of the name for a DeniablePermission is the name of the given permission (for example, "exit", "setFactory", "print.queueJob", etc). The naming convention follows the hierarchical property naming convention. An asterisk may appear by itself, or if immediately preceded by a "." may appear at the end of the name, to signify a wildcard match. For example, "*" and "java.*" are valid, while "*java", "a*b", and "java*" are not valid.

The rules for the second part of the name (things denied) are as follows. The second part of the name consists of comma separated resource names that should be denied access. The individual resource names are called denial strings These denial strings may be indiviually listed resources ("exit", "setFactory", "print.queueJob", etc). The individual denied strings may end with ".*" in order to signify a wildcard match. Examples of valid names with denial strings are

- "com.acme.*-com.acme.security.*"
- "*-com.acme.accounting.*,com.acme.cfo.getMail"

There are other rules about DeniablePermission names:

- A name may not have a '-' character if the grant portion of the name is not a wildcard
- A denial string may not be "*"
- A denial string may not have an embedded "*"
- The last character of a name may not be '-'

See Also:
Serialized Form

Constructor Summary

`DeniablePermission(String name)` This cuts off anything prior to the '-' character for the super
`DeniablePermission(String name, String actions)` This cuts off anything prior to the '-' character for the super

Method Summary

`boolean`	`equals(Object cmp)`
`int`	`hashCode()`

boolean	`implies(Permission p)`
String	`toString()`

Methods inherited from class java.security.BasicPermission
getActions, newPermissionCollection

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

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

Constructor Detail

6.1.1 DeniablePermission

```
public DeniablePermission(String name)
```

This cuts off anything prior to the '-' character for the super

Parameters:

`name` - The name with the optional '-' character

6.1.2 DeniablePermission

```
public DeniablePermission(String name,
                          String actions)
```

This cuts off anything prior to the '-' character for the super

Parameters:

`name` - The name with the optional '-' character

`actions` - The actions

Method Detail

6.1.3 implies

```
public boolean implies(Permission p)
```

Overrides:

`implies` in class `BasicPermission`

See Also:

`BasicPermission.implies(java.security.Permission)`

6.1.4 hashCode

```
public int hashCode()
```

All Page Within This Box

Overrides:
`hashCode` in class `BasicPermission`
See Also:
`BasicPermission.hashCode()`

6.1.5 equals

`public boolean **equals**(Object cmp)`

Overrides:
`equals` in class `BasicPermission`
See Also:
`BasicPermission.equals(java.lang.Object)`

6.1.6 toString

`public String **toString**()`

Overrides:
`toString` in class `Permission`

http://www-beace/beace-site/5.0/com.bea.sandbox.dp/javadoc/com/bea/sandbox/security/permission/DeniablePermission.html - skip-navbar_bottom#skip-navbar_bottom

Package **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS **FRAMES** **NO FRAMES** **All Classes** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Documentation is available at
http://docs-stage/msa/docs30
Copyright 2007 BEA Systems Inc.

6.2 Deny Permission Column

In this solution a deny permission column was added to the tuple table. There are several reasons this solution was reject. This solution did not handle certain important use cases (which will be detailed in section 6.2.10). Furthermore it is difficult to optimize this solution since all rows of the table must be evaluated even if one of the rows has returned an "allows" result.

The following are the discussions and requirements of that solution.

6.2.1 Add Deny Column to Permission Table Discussion

The first part of the solution is to take the security table which currently has three columns and add a fourth column. Currently, the three columns would be labeled "name", "condition" and "permission". In this solution the columns of the table would be labeled "name", "condition", "allows permission" and "denies permission".

Today a "permission" cell in the table contains a list of Permission objects. If any of these objects "imply" the permission being sought then the result of the security check is to allow the operation. Hence, the individual permissions act like a logical "OR" operation, since if *any* of the Permission objects in the cell return "true" then the security check returns "true" (or ALLOW). Note that at least one Permission object must "imply" the permission. Hence an empty permission set in the "permission" column of the table would always return "false" (or DENY).

In this solution the “allows permission” cell behaves in the same way as above, and would return the same “true” (ALLOW) or “false” (ABSTAIN) decision as it does currently. However, after the “allow permission” cell has been consulted the “denies permission” cell will also be checked. The “denies permission” cell must return “false” in order for the security check to pass. In other words the security check result can be gotten using the following pseudo-formula:

Security-check-result = (allow-permission-result) && !(denies-permission-result).

The “denies-permission” cells contains a list of Permission objects, just as the “allows-permission” cells do. The individual permissions in the “denies-permission” cells are OR’d together to get the result. Therefore in order for the whole security check to pass ALL of the Permissions in the “denies-permission” cell must NOT imply the Permission being asked for.

A simple example should make this clear. Assume I wish to grant access to all packages except the “com.acme.secret” package and the “com.acme.sauce” package. Here is a row of cells in the security table:

Name	**Condition**	**Allow Permission**	**Deny Permission**
Example	<empty> (always true)	Package(“*”)	Package(“com.acme.secret”), Package(“com.acme.sauce”)

If a bundle is trying to access the package “com.acme.transaction” they would first check against Package(“*”) which would return “true”. Then the system would check both Package(“com.acme.secret”) which would return “false” and Package(“com.acme.sauce”) which would return” false”, and so the “Deny Permission” cell would have a value of “false”. The security check: (allow-permission-result) && !(denies-permission-result) would end up being (true) && !(false) and would hence be “true” and access to the “com.acme.transaction” package would be allowed.

On the other hand, if a bundle is trying to access the package “com.acme.secret” they would first check against Package(“*”) which would return “true”. Then the system would check Package(“com.acme.secret”) which would return “true”, and so the “Deny Permission” cell would have a value of “true”. The security check (allow-permission-result) && !(denies-permission-result) would end up being (true) && !(true) and would hence be “false” and access to the “com.acme.secret” package would be denied.

Some things about this proposed solution:

1. The way the table works today if the permission succeeds against ANY row in the table then the check will succeed. Hence you could have a scenario where something is denied in one row of the table but succeeds in another row of the table. According to the current rules that would allow access to that permission. Specifying a separate “deny” table (rather than adding a column to the existing table) has been considered.

2. In the condition column an empty cell implies “true”. However, in the “Deny Permission” column it appears that an empty cell should imply “false”.

3. For this RFC to be complete new API and a new “normalized” file format will need to be defined.

All Page Within This Box

6.2.1.1 Deny Column Requirements

A fourth column shall be added to the conceptual permission table. For the purposes of these requirements the existing column in the tuple table will be called the “allow permissions” column while the new column being added will be called the “deny permissions” column.

The “deny permission” column of the table shall contain a list of PermissionInfo objects. A target permission shall be implied when the following are true:

1. At least one permission in the “allow permission” column has a permission that implies the target permission

2. None of the permissions in the “deny permission” column has a permission the implies the target permission

If there are no permissions in the “deny permission” column then the column naturally returns “false” and the result of the permission check comes solely from the permissions in the “allow permission” column.

The “deny permission” column shall be consulted when determining if a tuple should be postponed. Therefore the boxes in Figure 9.42 of the Conditional Permission Admin Specification that ask if a permission implies P shall take the “deny permission” column into account as described above.

However, due to these requirements it is no longer the case that any tuple which eventually returns “true” means that the permission check can succeed. All of the tuples whose conditions match and who have at least one entry in the “deny permissions” column shall be checked in order to ensure that no denial is present.

6.2.2 NOT Condition Discussion

Using the above specified mechanism it is possible to deny access to resources. While this achieves the requirements to deny resources there is still something missing. For example, suppose we wanted to allow bundles signed by the ACME Company to have access to all packages but we wanted all other bundles in the system to not have access to the “com.acme.secret” and “com.acme.sauce” packages. There is no way using standard OSGi conditions to allow this to happen. Instead, what is needed is a NOT boolean condition operator.

Consider the use case where you want ACME (the provider of the system software) to have access to all packages but everyone else should not be allowed access to the “com.acme.secret” and “com.acme.sauce” packages. In this case, you would need the Not condition. Your permission table would look something like this:

Name	Condition	Allows Permission	Deny Permission
ACME	Signed By ACME	Package(“*”)	<empty>

Non-System	NOT Signed By ACME	Package(“*”)	Package(“com.acme.secret”)
			Package(“com.acme.sauce”)

In the above table the bundles signed by ACME will have access to all packages, while all bundles NOT signed by Acme can access neither the “com.acme.secret” package nor the “com.acme.sauce” package.

Another example will illustrate how you could allow multiple providers of system software to use the deny permission feature. For example, the ACME Corporation may be using Spring to provide dependency injection. In this example, both ACME and Spring need to have access to all packages, but no-one else should be granted access to the same restricted packages as above. In this case, your permission table would look something like this:

Name	Condition	AllowsPermission	DenyPermission
ACME	Signed By ACME	Package(“*”)	<empty>
Spring	Signed By Spring	Package(“*”)	<empty>

Non-System	NOT Signed By ACME
	NOT Signed By Spring
		Package(“*”)	Package(“com.acme.secret”)
			Package(“com.acme.sauce”)

Since the Conditions in a single tuple row must ALL satisfy the condition the above table would achieve the desired result or allowing the system bundles (signed by Acme and Spring) to have access to all packages while denying access to the “com.acme.secret” and “com.acme.sauce” packages to all non-system bundles.

The interesting thing about the above examples is they can both be simplified with the following table:

Name	Condition	Allow Permission	Deny Permission
Default	<empty>	Package(“*”)	<empty>

Non-System	NOT Signed By ACME
	NOT Signed by Spring
		<empty>	Package(“com.acme.secret”)
			Package(“com.acme.sauce”)

The above table will achieve the same results as the previous table with fewer entries. The system may choose to optimize these sorts of conditions.

6.2.3 NotCondition Requirements

A new class named “org.osgi.service.condpermadmin.NotCondition” shall be added to the framework. The javadoc shall be considered a binding part of this specification, and is found in the next sub-section.

In particular, the NotCondition shall have another condition upon which it bases its return values. This base condition will be called the “base condition.” The NotCondition shall return the same values as the base for the mutable and postponed properties and shall return the logical NOT of the base satisfied property.

Currently the system understands the static form:

static Condition getCondition(Bundle bundle, String parameters);

However, the NotCondtion shall have the static factory method:

static Condition getCondition(Bundle bundle, Condition base);

The Encoded ConditionInfo string shall be enhanced to understand the “!” character at the start of the condition. If the “!” character is found the base condition shall be constructed as before, but it will be used as the base condition of the NotCondition. The encoded string shall have the form:

"[" + "!"* + fully-qualified-condition-class + " \"" + argument-string + "\"]"

Note that the "!" character is optional (denoted by the *).

Some examples of the encoded ConditionInfo string might include:

[!org.osgi.service.condpermadmin.BundleSignerCondition "* ; cn=Whatever, o=ACME, c=US"]

[!org.osgi.service.condpermadmin.BundleLocationCondition "/home/acme/foo/b.jar"]

6.2.4 NotCondition Javadoc

C:\tmp\osgi\rfc120\org\osgi\service\condpermadmin\NotCondition.html - skip-navbar_top#skip-navbar_top

Package **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS **FRAMES** **NO FRAMES** **All Classes** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

org.osgi.service.condpermadmin

Class NotCondition

```
java.lang.Object
  └org.osgi.service.condpermadmin.NotCondition
```

```
public class NotCondition
extends java.lang.Object
```

This is a condition that is formed as the logical NOT operation of another condition.

Method Summary

`static Condition`	`getCondition(Bundle bundle, Condition condition)` Creates a condition that returns the logical NOT of another condition.

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Method Detail

getCondition

```
public static Condition getCondition(Bundle bundle,
                                     Condition condition)
```

This uses the Condition form as proposed in RFC 120

Parameters:

`bundle` - The bundle for which this condition is being created

`condition` - The condition to "NOT"

Returns:

All Page Within This Box

A condition that will have the same mutable and postponed values as the base condition but which will return the logical NOT of the satisfied value of the base condition.

C:\tmp\osgi\rfc120\org\osgi\service\condpermadmin\NotCondition.html - skip-navbar_bottom#skip-navbar_bottom

Package **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS   **FRAMES** **NO FRAMES** **All Classes** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD   DETAIL: FIELD | CONSTR | METHOD

6.2.5 Other API modifications

In order to fully support the modifications described in sections 6.2.1 and 6.2.2 some existing API's will also need to change.

The following sections will give javadoc for the new API and constructors and should be considered a binding part of this specification.

6.2.6 org.osgi.service.condpermadmin.ConditionInfo

The following API shall be added to the org.osgi.service.condpermadmin.ConditionInfo class:

6.2.6.1 Constructor

ConditionInfo

```
public ConditionInfo(boolean not,
                     java.lang.String type,
                     java.lang.String[] args)
```

Constructs a `ConditionInfo` from the specified type and args.

Parameters:

`not` - If true, the condition described should return the associated NotCondition, using the type and args as the base Condition.

`type` - The fully qualified class name of the Condition represented by this `ConditionInfo`.

`args` - The arguments for the Condition. These arguments are available to the newly created Condition by calling the `getArgs()` method.

Throws:

`java.lang.NullPointerException` - If `type` is `null`.

6.2.6.2 isNot method

isNot

```
public boolean isNot()
```

Returns true if this ConditionInfo describes a NotCondition

Returns:

true if this ConditionInfo describes a NotCondition

6.2.7 org.osgi.service.condpermadmin.ConditionPermissionInfo

The following API shall be added to the org.osgi.service.condpermadmin.ConditionPermissionInfo interface.

6.2.7.1 getDenyPermissionInfos method

getDenyPermissionInfos

```
PermissionInfo[] getDenyPermissionInfos()
```

Returns the Permission Infos for the Deny Permission column in this Condition Permission Info.

Returns:

The Deny Permission Infos for the Permission in this Conditional Permission Info.

6.2.8 org.osgi.service.condpermadmin.ConditionalPermissionAdmin

The following API shall be added to the org.osgi.service.condpermadmin.ConditionPermissionAdmin interface.

6.2.8.1 addConditionalPermissionInfo

addConditionalPermissionInfo

```
ConditionalPermissionInfo addConditionalPermissionInfo(ConditionInfo[] conds,
                                                       PermissionInfo[] perms,
                                                       PermissionInfo[] denyPerms)
```

Create a new Conditional Permission Info. The Conditional Permission Info will be given a unique, never reused name.

Parameters:

`conds` - The Conditions that need to be satisfied to enable the corresponding Permissions.

`perms` - The Permissions that are enable when the corresponding Conditions are satisfied.

`denyPerms` - The Permissions that are denied when the corresponding Conditions are satisfied.

Returns:

The ConditionalPermissionInfo for the specified Conditions and Permissions.

Throws:

`java.lang.SecurityException` - If the caller does not have `AllPermission`.

6.2.8.2 setConditionalPermissionInfo

setConditionalPermissionInfo

```
ConditionalPermissionInfo setConditionalPermissionInfo(java.lang.String name,
                                                       ConditionInfo[] conds,
                                                       PermissionInfo[] perms,
                                                       PermissionInfo[] denyPerms)
```

Set or create a Conditional Permission Info with a specified name. If the specified name is `null`, a new Conditional Permission Info must be created and will be given a unique, never reused name. If there is currently no Conditional Permission Info with the specified name, a new Conditional Permission Info must be created with the specified name. Otherwise, the Conditional Permission Info with the specified name must be updated with the specified Conditions and Permissions.

Parameters:

`name` - The name of the Conditional Permission Info, or `null`.

`conds` - The Conditions that need to be satisfied to enable the corresponding Permissions.

`perms` - The Permissions that are enable when the corresponding Conditions are satisfied.

`denyPerms` - The Permissions that are denied when the corresponding Conditions are satisfied.

Returns:

The ConditionalPermissionInfo that for the specified name, Conditions and Permissions.

Throws:

`java.lang.SecurityException` - If the caller does not have `AllPermission`.

6.2.9 Issues

6.2.10 Friends API

One use case (which is not in the original set of use cases) that is not handled cleanly by the above solution is the friends use case. In this use case we want to express the following scenario:

4. Pepsi wants to deny permissions to com.pepsi.* for everyone but Pepsi

5. Pepsi wants to allow Coke to have permission to com.pepsi.z.*

Using the existing scheme you cannot express the above set of requirements simply. The best you could do would be to have something like this:

Name	**Condition**	**Allow Permissions**	**Deny Permissions**
N1	! Signed by Pepsi ! Signed by Coke	Package(*)	Package(“com.pepsi.*”)

N2	Signed by Coke	Package(*)	Package(“com.pepsi.a.*”)
			Package(“com.pepsi.b.*”)
			…
			Package(“com.pepsi.y.*”)

N3	<empty>	Package(*)	<empty>

Notice that in the N2 row above that it denies access to Coke to every package **except** the com.pepsi.z.* package. This is brittle and error prone, since with every new package hierarchy (perhaps com.pepsi.aa.*) that Pepsi might add to their system a new Permission would have to be added in the N2 line. It is this sort of complexity that this solution was attempting to address.

Given that this not one of the original use-cases and there does not appear to be a satisfactory solution to this problem this specification will not address this use case.

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-78 Security Use Cases

8.2 Author's Address

Name	John Wells
Company	BEA Systems, Inc.
Address	150 Allen Road, Liberty Corner, NJ
Voice	(908) 580-3127
e-mail	jwells@bea.com

8.3 Acronyms and Abbreviations

8.4 End of Document