Java™ Authorization Contract for Containers

Please send comments to: jsr-115-comments@jcp.org

JSR-115

Java Community Process℠ (JCP℠) 2.1

Maintenance Release 4

Version 1.1

Specification Lead: Ron Monzillo
Sun Microsystems, Inc.

Specification: JSR-000115 Java(tm) Authorization Contract for Containers ("Specification")
Version: 1.1
Status: Maintenance Release 4
Release: 8 May 2006

LIMITED LICENSE GRANTS

1. License for Evaluation Purposes. Sun hereby grants you a fully-paid, non-exclusive, non-transferable, worldwide, limited license (without the right to sublicense), under Sun's applicable intellectual property rights to view, download, use and reproduce the Specification only for the purpose of internal evaluation. This includes (i) developing applications intended to run on an implementation of the Specification, provided that such applications do not themselves implement any portion(s) of the Specification, and (ii) discussing the Specification with any third party; and (iii) excerpting brief portions of the Specification in oral or written communications which discuss the Specification provided that such excerpts do not in the aggregate constitute a significant portion of the Specification.

2. License for the Distribution of Compliant Implementations. Sun also grants you a perpetual, non-exclusive, non-transferable, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights or, subject to the provisions of subsection 4 below, patent rights it may have covering the Specification to create and/or distribute an Independent Implementation of the Specification that: (a) fully implements the Specification including all its required interfaces and functionality; (b) does not modify, subset, superset or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required/authorized by the Specification or Specifications being implemented; and (c) passes the Technology Compatibility Kit (including satisfying the requirements of the applicable TCK Users Guide) for such Specification ("Compliant Implementation"). In addition, the foregoing license is expressly conditioned on your not acting outside its scope. No license is granted hereunder for any other purpose (including, for example, modifying the Specification, other than to the extent of your fair use rights, or distributing the Specification to third parties). Also, no right, title, or interest in or to any trademarks, service marks, or trade names of Sun or Sun's licensors, Sun or the Sun's licensors is granted hereunder. Java, and Java-related logos, marks and names are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

3. Pass-through Conditions. You need not include limitations (a)-(c) from the previous paragraph or any other particular "pass through" requirements in any license You grant concerning the use of your Independent Implementation or products derived from it. However, except with respect to Independent Implementations (and products derived from them) that satisfy limitations (a)-(c) from the previous paragraph, You may neither: (a) grant or otherwise pass through to your licensees any licenses under Sun's applicable intellectual property rights; nor (b) authorize your licensees to make any claims concerning their implementation's compliance with the Spec in question.

4. Reciprocity Concerning Patent Licenses.

a. With respect to any patent claims covered by the license granted under subparagraph 2 above that would be infringed by all technically feasible implementations of the Specification, such license is conditioned upon your offering on fair, reasonable and non-discriminatory terms, to any party seeking it from You, a perpetual, non-exclusive, non-transferable, worldwide license under Your patent rights which are or would be

infringed by all technically feasible implementations of the Specification to develop, distribute and use a Compliant Implementation.

b. With respect to any patent claims owned by Sun and covered by the license granted under subparagraph 2, whether or not their infringement can be avoided in a technically feasible manner when implementing the Specification, such license shall terminate with respect to such claims if You initiate a claim against Sun that it has, in the course of performing its responsibilities as the Specification Lead, induced any other entity to infringe Your patent rights.

c. Also with respect to any patent claims owned by Sun and covered by the license granted under subparagraph, where the infringement of such claims can be avoided in a technically feasible manner when implementing the Specification such license, with respect to such claims, shall terminate if You initiate a claim against Sun that its making, having made, using, offering to sell, selling or importing a Compliant Implementation infringes Your patent rights.

5. Definitions. For the purposes of this Agreement: "Independent Implementation" shall mean an implementation of the Specification that neither derives from any of Sun's source code or binary code materials nor, except with an appropriate and separate license from Sun, includes any of Sun's source code or binary code materials; "Licensor Name Space" shall mean the public class or interface declarations whose names begin with "java", "javax", "com.sun" or their equivalents in any subsequent naming convention adopted by Sun through the Java Community Process, or any recognized successors or replacements thereof; and "Technology Compatibility Kit" or "TCK" shall mean the test suite and accompanying TCK User's Guide provided by Sun which corresponds to the Specification and that was available either (i) from Sun 120 days before the first release of Your Independent Implementation that allows its use for commercial purposes, or (ii) more recently than 120 days from such release but against which You elect to test Your implementation of the Specification.

This Agreement will terminate immediately without notice from Sun if you breach the Agreement or act outside the scope of the licenses granted above.

DISCLAIMER OF WARRANTIES

THE SPECIFICATION IS PROVIDED "AS IS". SUN MAKES NO REPRESENTATIONS OR WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT (INCLUDING AS A CONSEQUENCE OF ANY PRACTICE OR IMPLEMENTATION OF THE SPECIFICATION), OR THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE. This document does not represent any commitment to release or implement any portion of the Specification in any product. In addition, the Specification could include technical inaccuracies or typographical errors.

LIMITATION OF LIABILITY

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUE, PROFITS OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED IN ANY WAY TO YOUR HAVING, IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION, EVEN IF SUN AND/OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You will indemnify, hold harmless, and defend Sun and its licensors from any claims arising or resulting from: (i) your use of the Specification; (ii) the use or distribution of your Java application, applet and/or

implementation; and/or (iii) any claims that later versions or releases of any Specification furnished to you are incompatible with the Specification provided to you under this license.

RESTRICTED RIGHTS LEGEND

U.S. Government: If this Specification is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in the Software and accompanying documentation shall be only as set forth in this license; this is in accordance with 48 C.F.R. 227.7201 through 227.7202-4 (for Department of Defense (DoD) acquisitions) and with 48 C.F.R. 2.101 and 12.212 (for non-DoD acquisitions).

REPORT

If you provide Sun with any comments or suggestions concerning the Specification ("Feedback"), you hereby: (i) agree that such Feedback is provided on a non-proprietary and non-confidential basis, and (ii) grant Sun a perpetual, non-exclusive, worldwide, fully paid-up, irrevocable license, with the right to sublicense through multiple levels of sublicensees, to incorporate, disclose, and use without limitation the Feedback for any purpose.

GENERAL TERMS

Any action related to this Agreement will be governed by California law and controlling U.S. federal law. The U.N. Convention for the International Sale of Goods and the choice of law rules of any jurisdiction will not apply.

The Specification is subject to U.S. export control laws and may be subject to export or import regulations in other countries. Licensee agrees to comply strictly with all such laws and regulations and acknowledges that it has the responsibility to obtain such licenses to export, re-export or import as may be required after delivery to Licensee.

This Agreement is the parties' entire agreement relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, conditions, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification to this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

Rev. January, 2006

Sun/Final/Full

Contents

Preface

Status of Document

This document is the Final Release of the Java™ Authorization Contract for Containers Version 1.0 specification and represents the definition of this technology as implemented by the reference implementation (RI) and verified by the technology compatibility kit (TCK) . This specification was developed under the Java Community Process (JCP2.1).

Audience

This document is intended for developers of the RI and TCK and for those who will be delivering implementations of this technology in their products.

Abstract

This specification defines new java.security.Permission classes to satisfy the Java EE authorization model. The specification defines the binding of container access decisions to operations on instances of these permission classes. The specification defines the semantics of policy providers that employ the new permission classes to address the authorization requirements of Java EE, including the following:

- the definition of roles as named collections of permissions
- the granting to principals of permissions corresponding to roles
- the determination of whether a principal has been granted the permissions of a role (e.g. isCallerInRole)
- the definition of identifier to role mappings that bind application embedded identifiers to application scoped role names.

The specification defines the installation and configuration of authorization providers for use by containers. The specification defines the interfaces that a

provider must make available to allow container deployment tools to create and manage permission collections corresponding to roles.

Keywords

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC2119 [KEYWORDS].

Acknowledgements

This draft of the specification incorporates the contributions of the RI and TCK teams with the output of the JSR115 Expert Group. The JSR 115 Expert Group included the following members:

Steven Bazyl	RSA Security, Inc.
Sean Dolan	Hitachi Computer Products
Herb Erickson	SilverStream Software
Gary Ellison	Sun Microsystems
Neil Forrest	Dyoti Enterprises Ltd
Johan Gellner	Tmax Soft, Inc.
Craig Heath	Individual
Hal Lockhart	Entegrity Solutions
Larry McCay	Hewlett-Packard Company
Serge Mister	Entrust, Incorporated
Ron Monzillo	Sun Microsystems
Anthony Nadalin	Tivoli Systems, Incorporated
Nataraj Nagaratnam	International Business Machines Corporation
Vijakumar Natarajan	Borland Software Corporation
Raymond K. Ng	Oracle Corporation
Samir Nigam	Sybase, Incorporated
Henry Pasternack	Netegrity, Incorporated
Paul Patrick	BEA Systems
Francis Pouatcha	Individual
Jyri Virkki	iPlanet

The RI, the TCK, and the improvements to the specification made as a result of the experiences gained during these actvities are the result of the fine work of the following individuals:

Jean-Francois Arcand	Sun Microsystems
Carla Carlson	Sun Microsystems

Shing Wai Chan	Sun Microsystems
Eric Jendrock	Sun Microsystems
Jagadesh Babu Munta	Sun Microsystems
Tony Ng	Sun Microsystems
Raja Perumal	Sun Microsystems
Deepa Singh	Sun Microsystems
Harpreet Singh	Sun Microsystems

The following people are among many who commented on the specification, and in so doing, contributed to its final form. I would like to recognize the contributions of everyone who commented on the specification.

Rajeev Angal	iPlanet
Lambert Boskamp	SAP AG
William Cox	BEA Systems
Paul Ferwerda	BEA Systems
Charlie Lai	Sun Microsystmes
Rosanna Lee	Sun Microsystems
Robert Naugle	Hewlett-Packard Company
Bob Scheifler	Sun Microsystems
Bill Shannon	Sun Microsystems
Neil Smithline	BEA Systems
Sirish Vepa	Sybase, Incorporated
Kai Xu	Sun Microsystems

CHAPTER 1

Overview

This specification defines a contract between Java EE containers and authorization policy modules such that container authorization functionality can be provided as appropriate to suit the operational environment.

1.1 Introduction

The contract defined by this specification is divided into three subcontracts. Taken together, these subcontracts describe the installation and configuration of authorization providers such that they will be used by containers in performing their access decisions. The three subcontracts are the Provider Configuration Subcontract, the Policy Configuration Subcontract, and the Policy Decision and Enforcement Subcontract.

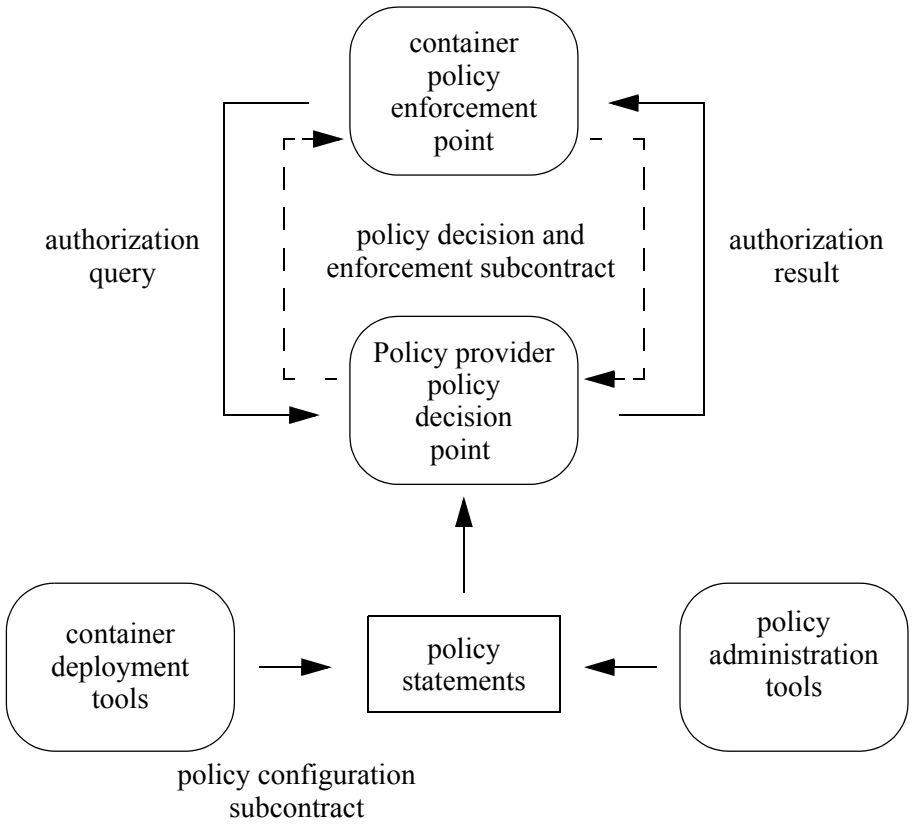


Figure 1.1 Policy Configuration and Enforcement Subcontracts

1.2 Terminology

Java EE application

A collection of Java EE modules that must share a common principal-to-role-mapping

deploy (e.g. an application)

The sequence of operations comprised of *completing the declaration* of an application or module's dependencies on container facilities, *binding the declared dependencies* to specific mechanisms or features of an operational environment, *installing or distributing* the application

software and related configuration information to one or more application servers, and *activating* the software such that it is available to service requests.

undeploy (e.g. an application)

The combined process of *stopping* an application and then *removing* the software and configuration information corresponding to the application or a module of the application from one or more application servers.

redeploy (e.g. a module of an application)

The *repackaging* of an application or module to accommodate modification of implementation and or of declared dependencies and or of the binding of declared dependencies to mechanisms, combined with *undeploying* a corresponding module or application, followed by *redistribution* and *activation* of the modified software and or configuration.

grant

The act of assigning to entities the right to perform a set of activities that is the subject of an authorization decision (that is, a permission).

hostname

The name of a logical host of an application server, as may be used in the composition of a servlet policy context identifier.

JAAS Policy interface

The `javax.security.auth.Policy` abstract class.

JAAS Policy provider

An instance of a class that implements the JAAS Policy interface.

permission

Represents a set of activities (a set of one or more operations on some set of one or more resources) that is the target of an authorization decision.

Policy Context

The collection of policy statements within a policy provider that effect access to the resources of one or more deployed modules.

Policy Context Identifier

A unique string value that identifies the collection of policy statements corresponding to a policy context within a policy provider.

policy statement

A representation of the circumstances under which the set of activities represented by a permission are to be authorized.

excluded policy statement

A representation of the decision not to authorize a set of activities represented by a permission independent of factors that might otherwise effect the outcome of the decision.

unchecked policy statement

A representation of the decision to authorize a set of activities represented by a permission independent of factors that might otherwise effect the outcome of the decision.

principal

[1] (Java) A security attribute acquired as a result of authentication by entities that perform activities.

[2] An entity that performs activities.

principal-to-role mapping

The act of granting to principals the set of permissions that comprise a role.

privilege

A security attribute that may be assigned to entities and that may be used to differentiate an entity's right to perform activities.

Provider

The software component that contains implementations of the policy configuration, and policy decision classes defined by this specification.

reference-to-role mapping

The component-scoped transformation of component embedded role aliases into application-scoped role names. The transformation is defined at application deployment and perhaps modified by policy management.

role

[1] A named set of permissions that may be granted to principals.

[2] A principal that has been granted permissions or that is used as a privilege.

1.3 Assumptions

1. The contract defined in this JSR must be applicable to both J2EE 1.3 platforms and J2EE 1.4 platforms and to future versions of the Java EE platform.
2. We are defining a contract to be satisfied by "standard" Java SE Policy providers. In a J2EE 1.3 context we expect the provider to include a JAAS Policy provider.
3. J2EE 1.4 platforms will be required to implement the contract defined by this JSR. This contract will be a required element of subsequent versions of the Java EE platform.
4. Support for the contract by J2EE 1.3 platforms is optional. It is expected that there may be aspects of the contract that are Java EE release specific.
5. We do not expect to add new decision interfaces to Policy (JAAS or Java SE) to accomplish the work of this JSR.
6. Java EE application roles will be modeled as collections of permissions that are granted to principals.
7. A principal that is in a role is granted all the permissions of the collection. However, the converse is not true. That is, a principal that has been granted all the permissions of a role is not necessarily in the role (as determined by isCallerInRole()).

8. This contract will shift the responsibility for performing all of the authorization decisions pertaining to a Java EE application to the policy provider. Accordingly, the following mappings will become the responsibility of the provider.

    - permissions to roles
    - principals to roles
    - (Application embedded) role references to role names

9. It is assumed that there are providers that are unable to enumerate all the permissions that pertain to a subject/protection domain before returning from Policy.getPermissions().

10. Any interfaces that this contract defines to be used by containers and or container deployment tools to create policy statements within a policy provider must be compatible with a module-at-a-time application deployment mechanism.

11. Where the Enterprise JavaBeans™ (EJB™) or Java Servlet™ specifications are incomplete or ambiguous in their specification of authorization functionality, the contract defined in this document may require additional semantics. Additional or clarifying semantics will only be adopted by this specification based on their acceptance by the spec lead and expert group of the corresponding component specification.

12. The EJB policy decisions performed by providers may require access to the arguments of the EJB invocation and or (for entity beans) the container managed EJB instance state.

1.4 Requirements

1. This contract must support providers that are unable to determine, before returning from Policy.getPermissions(), all the permissions that pertain to a subject/protection domain.

2. Each Policy provider that satisfies this contract must perform or delegate to another provider all the permission evaluations requested via its interface in the JRE; not just those made by the container to implement Java EE security functionality.

3. Each provider must export interfaces (defined by this contract) for use by containers and or container deployment tools to create policy statements within the policy store of the provider. These interfaces must be used when an application or module is deployed in a container.

4. Each provider must satisfy all of the authorization requirements of the EJB and Servlet specifications corresponding to the target platform.

5. The evaluation of a permission corresponding to a resource must identify the context of the resource's use such that different policy can be applied to a resource used in different contexts (that is, applications or instances of an application).

6. In the case of Servlet resources, the provider must be able to associate a distinct policy context with each context root (including context roots created to support virtual hosting) hosted by the server.

7. In protecting Servlet resources, a provider must select the policy statements that apply to a request according to the constraint matching and servlet mapping rules defined by the Servlet specification.

8. To support this contract in a Servlet environment, a container or its deployment tools must create policy statements as necessary to support Servlet's "default role-ref semantic".

9. For a container to support this contract, it must execute in an environment controlled by a Java SE SecurityManager. Containers may also execute in environments that are not controlled by a Java SE SecurityManager. Section 1.5, "Running Without a SecurityManager" defines changes to this contract that apply to containers running without a Java SE SecurityManager.

10. Policy providers must perform the permission evaluations corresponding to container pre-dispatch decisions and application embedded privilege tests (i.e isUserInRole and isCallerInRole) without requiring that containers establish particular values for any of the non-principal attributes of the one or more java.security.ProtectionDomain objects that are the subject of the evaluation.

1.4.1 Non Requirements

1. This JSR does not require that containers support server-side authentication module plug-ins for the purpose of populating subjects with authorization provider specific principals.

2. This JSR does not require that subjects be attributed with role principals as a result of authentication.

3. This JSR does not define or mandate a specific policy language to be used by providers. Each provider must define its own syntax, mechanisms, and administrative interfaces for granting permissions to principals.

4. The JSR does not require that providers support a policy syntax for granting to principals roles as collections of permissions.

5. Although the JSR is focused on defining permissions and policy for use by Java EE containers, we make no restrictions on the use of this information by other containers or applications, or on support by containers or providers of other permissions or policy.

6. It is not the intent of this JSR to extend or modify the Java EE authorization model to be equivalent to standard RBAC models for access control.

1.5 Running Without a SecurityManager

The following list defines changes to this contract that apply to containers running without a Java SE SecurityManager.

1. The restrictions defined in Section 3.3, “Permission to Configure Policy" need not be enforced. Also, the containers of the application server must not be denied permission to perform any operation that would have been permitted in the presence of a SecurityManager.

2. Such containers are not required (before dispatching a call) to associate an AccessControlContext with the call thread (as otherwise required by Section 4.1.3, “Pre-dispatch Decision" and Section 4.3.1, “EJB Pre-dispatch Decision").

3. When performing the operations defined in Section 4.7, “Checking AccessControlContext Independent Grants" and in Section 4.8, “Checking the Caller for a Permission", such containers must not employ the SecurityManager.checkPermission and AccessControlContext.checkPermission techniques defined in these sections.

CHAPTER 2

Provider Configuration Subcontract

The Provider Configuration Subcontract defines the requirements placed on providers and containers such that Policy providers may be integrated with containers.

2.1 Policy Implementation Class

The contract defined by this specification has been designed to work in J2SE 1.4 or later Java Standard Edition environments with the default `java.security.Policy` implementation class, and in J2SE 1.3 environments with the default `javax.security.auth.Policy` implementation class. Support for the contract defined by this specification is optional in J2EE 1.3 environments.

Java platforms provide standard security properties whose values may be defined to cause replacement of the default system Policy implementation classes. The security property, "`policy.provider`", may be used to replace the default `java.security.Policy` implementation class. Similarly, the security property, "`auth.policy.provider`", may be used to replace the default `javax.security.auth.Policy` implementation class. These properties are defined in the Java security properties file, and replacement is accomplished by setting their value to the fully qualified name of the desired Policy implementation class. The contract defined in this specification, is dependent on the Policy replacement mechanisms of the target Java environment. An application server that supports this contract must allow replacement of the top level

java.security.Policy object used by every JRE of the containers of the application server.

2.2 Permission Implementation Classes

This contract defines a Java standard extension package, `javax.security.jacc`, that contains (among other things) Permission classes to be used by containers in their access decisions.

2.3 Policy Configuration Interface

The `javax.security.jacc` package defines an abstract factory class that implements a static method that uses a system property to find and instantiate a provider specific factory implementation class. The abstract factory class is `javax.security.jacc.PolicyConfigurationFactory`, the static method is `getPolicyConfigurationFactory`, and the system property is `javax.security.jacc.PolicyConfigurationFactory.provider`.

The abstract factory class also defines an abstract public method used to create or locate instances of the provider specific class that implements the interface used to define policy contexts within the associated Policy provider. The method is `getPolicyConfiguration` and the interface is `javax.security.jacc.PolicyConfiguration`.

The abstract `PolicyConfigurationFactory` class and the `PolicyConfiguration` interface are defined in Chapter "API," which begins on page 55. Use of the `PolicyConfiguration` interface is defined in Chapter "Policy Configuration Subcontract," which begins on page 17.

2.4 PolicyContext Class and Context Handlers

This `javax.security.jacc` package defines a utility class that is used by containers to communicate policy context identifiers to Policy providers. The utility class is `javax.security.jacc.PolicyContext`, and this class implements static methods that are used to communicate policy relevant context values from containers to Policy providers. Containers use the static method `PolicyContext.setContextID` to associate a policy context identifier with a thread on which they are about to call a decision interface of a Policy provider. Policy providers use the static method `PolicyContext.getContextID` to

obtain the context identifier established by a calling container. The role of policy context identifiers in access decisions is described in Section 3.1.1, "Policy Contexts and Policy Context Identifiers".

In addition to the methods used to communicate policy context identifiers, the `javax.security.jacc.PolicyContext` class also provides static methods that allow container specific context handlers that implement the `javax.security.jacc.PolicyContextHandler` interface to be registered with the `PolicyContext` class. The `PolicyContext` class also provides static methods that allow Policy providers to activate registered handlers to obtain additional policy relevant context to apply in their access decisions.

The `PolicyContext` utility class and the `PolicyContextHandler` interface are defined in Chapter "API," which begins on page 55. Use of the `PolicyContext` class is defined in Chapter "Policy Configuration Subcontract," which begins on page 17.

2.5 What a Provider Must Do

Each JRE of an application server must be provided with classes that implement the `PolicyConfigurationFactory` class and `PolicyConfiguration` interface. These classes must be compatible with the Policy implementation class installed for use by the JRE. In the case where the provider is not seeking to replace the Policy implementation used by the JRE, no other components need be provided.

If the provider is seeking to replace the Policy implementation used by the JRE, then the JRE must be provided with an environment specific Policy implementation class. If the JRE is running a J2SE 1.4 or later Java Standard Edition environment, then it must be provided with an implementation of the `java.security.Policy` class. If the JRE is running a J2SE 1.3 security environment, it must be provided with an implementation of the `javax.security.auth.Policy` class (that is, a JAAS Policy object).

A replacement Policy object must assume responsibility for performing all policy decisions within the JRE in which it is installed that are requested by way of the Policy interface that it implements. A replacement Policy object may accomplish this by delegating non-`javax.security.jacc` policy decisions to the corresponding default system Policy implementation class. A replacement Policy object that relies in this way on the corresponding default Policy implementation class must identify itself in its installation instructions as a "delegating Policy provider".

The standard security properties mechanism for replacing a default system Policy implementation (see Section 2.1, “Policy Implementation Class) should not be used to replace a default system Policy provider with a delegating Policy provider.

2.6 Optional Provider Support for JAAS Policy Object

In J2SE 1.4, the subject based authorization functionality of the JAAS Policy interface has been integrated into `java.security.Policy`, and the JAAS Policy interface (as a separate entity) has been deprecated. This does not mean that the JAAS Policy interface was removed, but rather that the essential parts of it have been tightly integrated into the J2SE 1.4 platform.

According to this contract, a J2SE 1.4 or later Java Standard Edition security environment may support replacement of the JAAS Policy object if and only if all javax.security.jacc policy decisions performed by the replacement JAAS Policy object return the same result as when the java.security.Policy interface is used. To satisfy this requirement, the replacement JAAS Policy object must be compatible with the implementations of `PolicyConfigurationFactory` and `PolicyConfiguration` interface provided for use with the java.security.Policy implementation class.

2.7 What the Application Server Must Do

An application server or container must bundle or install the `javax.security.jacc` standard extension. This package must include the abstract `javax.security.jacc.PolicyConfigurationFactory` class, the `javax.security.jacc.PolicyConfiguration` and `javax.security.jacc.PolicyContextHandler` interfaces, and implementations of the `javax.security.jacc.PolicyContextException` exception, `the javax.security.jacc` Permission classes, and the `javax.security.jacc.PolicyContext` utility class. The Permission classes of the `javax.security.jacc` package are:

- `javax.security.jacc.EJBMethodPermission`
- `javax.security.jacc.EJBRoleRefPermission`
- `javax.security.jacc.WebResourcePermission`
- `javax.security.jacc.WebRoleRefPermission`
- `javax.security.jacc.WebUserDataPermission`

To enable delegation of non-`javax.security.jacc` policy decisions to default system Policy providers, all application servers must implement the following Policy replacement algorithm. The intent of the algorithm is to ensure that Policy objects can capture the instance of the corresponding default system Policy object during their integration into a container and such that they may delegate non-container policy evaluations to it.

For each JRE of a J2EE 1.4 or later version Java EE application server, if the system property "`javax.security.jacc.policy.provider`" is defined, the application server must construct an instance of the class identified by the system property, confirm that the resulting object is an instance of `java.security.Policy`, and set, by calling the `java.security.Policy.setPolicy` method, the resulting object as the corresponding Policy object used by the JRE. For example:

```
String javaPolicy = System.getProperty(
    "javax.security.jacc.policy.provider"
);

if (javaPolicy != null) {
    try {
        java.security.Policy.setPolicy(
          (java.security.Policy)
            Class.forName(javaPolicy).newInstance()
        );
    } catch (ClassNotFoundException cnfe) {
        // problem with property value or classpath
    } catch (IllegalAccessException iae) {
        // problem with policy class definition
    } catch (InstantiationException ie) {
        // problem with policy instantiation
    } catch (ClassCastException cce) {
        // Not instance of java.security.policy
```

```
            }
        }
```

An application server that chooses to support this contract in a J2SE 1.3 environment must perform the policy replacement algorithm described above when the system property "`javax.security.jacc.auth.policy.provider`" is defined. That is, for each JRE of the application server, the server must construct an instance of the class identified by the system property, confirm that the resulting object is an instance of `javax.security.auth.Policy`, and set, by calling `javax.security.auth.Policy.setPolicy` method, the resulting object as the corresponding Policy object used by the JRE.

Once an application server has used either of the system properties defined in this section to replace a Policy object used by a JRE, the application server must not use setPolicy to replace the corresponding Policy object of the running JRE again.

The requirements of this section have been designed to ensure that containers support Policy replacment and to facilitate delegation to a default system Policy provider. These requirements should not be interpreted as placing any restrictions on the delegation patterns that may be implemented by replacement Policy modules.

2.7.1 Modifications to the JAAS SubjectDomainCombiner

The reference implementation of the combine method of the JAAS `SubjectDomainCombiner` returns protection domains that are constructed with a `java.security.Permissions` collection.This is the norm in J2SE 1.3 environments, and it also occurs in J2SE 1.4 and Java Standard Edition 5.0 environments when the installed JAAS Policy implementation class is not the `com.sun.security.auth.PolicyFile` class (that is, the JRE is operating in backward compatibility mode with respect to JAAS Policy replacement). The use of `java.security.Permissions` by the `SubjectDomainCombiner` forces JAAS Policy providers to compute all the permissions that pertain to a subject and code source and effectively precludes integration of Policy subsystems that are not capable of doing so. To ensure that the implementation of the JAAS `SubjectDomainCombiner` does not preclude integration of a class of Policy providers, this contract imposes the following requirement and recommendation on application servers.

To satisfy the contract defined by this specification, a J2EE 1.3 application server must install or bundle, such that it is used by every JRE of the application server, a `javax.security.auth.SubjectDomainCombiner` whose `combine` method returns protection domains constructed using the permission collections returned by `javax.security.auth.Policy.getPermisions`. It is recommended that this requirement also be satisfied by J2EE 1.4 and later version Java EE application servers in the case where `javax.security.auth.Policy` is used (in backward compatibility mode) to perform javax.security.jacc policy decisions.

CHAPTER 3

Policy Configuration Subcontract

The Policy Configuration Subcontract defines the interactions between container deployment tools and providers to support the translation of declarative Java EE authorization policy into policy statements within a Java SE Policy provider.

This subcontract also applies to the translation of authorization policy annotations that have an equivalent representation in Java EE deployment descriptor policy constructs (i.e., `security-constraint`, `method-permission`, `security-role-ref`, and `exclude-list` elements).

3.1 What a Java EE Platform's Deployment Tools Must Do

The `getPolicyConfigurationFactory` method must be used in every JRE to which the components of the application or module are being deployed to find or instantiate PolicyConfigurationFactory objects.

```
PolicyConfigurationFactory pcf =
    PolicyConfigurationFactory.getPolicyConfigurationFactory();
```

The `getPolicyConfiguration` method of the factories must be used to find or instantiate `PolicyConfiguration` objects corresponding to the application or modules being deployed.

```
String petContextID = "acme-pet-server /petstore";

PolicyConfiguration petPC =
   pcf.getPolicyConfiguration(petContextID,true);
```

The declarative authorization policy statements derived from the application or module deployment descriptor(s) must be translated to create instances of the corresponding `javax.security.jacc` Permission classes.

```
WebResourcePermission webPerm =
    new WebResourcePermission("/elephant","GET");
```

Methods of the `PolicyConfiguration` interface must be used with the permissions resulting from the translation to create policy statements within the `PolicyConfiguration` objects.

```
petPC.addToRole("customer",webPerm);
```

The `PolicyConfiguration` objects must be linked such that the same principal-to-role mapping will be applied to all the modules of the application.

```
petPC.linkConfiguration(petFoodPC);
```

The PolicyConfiguration objects must be placed in Service such that they will be assimilated into the Policy providers used by the containers to which the application has been deployed.

```
petPC.commit();
```

Independent of this specification, J2EE deployment tools must translate and complete the declarative policy statements appearing in deployment descriptors into a form suitable for securing applications on the platform. On versions of the Java EE platform that require support for authorization policy annotations, the deployment tools must combine policy annotations in Java code with policy statements appearing in deployment descriptors to yield complete representations of authorization policy suitable for securing applications on the platform. The rules for combining authorization policy annotations with declarative policy statements are described in the versions of the EJB, Servlet, and Java EE platform specifications that require support for the annotations. Independent of whether annotations factor in the translation, the resulting policy statements may differ in form from the policy statements appearing in the deployment descriptors. The policy translation defined by this subcontract is described assuming that the policy statement form used by a platform is identical to that used to express policy in the deployment descriptors. Where this is not the case, the output of the translation must be equivalent to the translation that would occur if policy was completely specified in the deployment descriptors and the translation had proceeded directly from the deployment descriptors to the Java SE policy forms defined by this subcontract. Two translations are equivalent if they produce corresponding collections of unchecked, excluded, and role permissions, and all of the

permissions of each such collection are implied by the permissions of the corresponding collection of the other translation.

3.1.1 Policy Contexts and Policy Context Identifiers

It must be possible to define separate authorization policy contexts corresponding to each deployed instance of a Java EE module. This per module scoping of policy context is necessary to provide for the independent administration of policy contexts corresponding to individual application modules (perhaps multiply deployed) within a common Policy provider.

Each policy context contains all of the policy statements (as defined by this specification) that effect access to the resources in one or more deployed modules. At policy configuration, a `PolicyConfiguration` object is created for each policy context, and populated with the policy statements (represented by permission objects) corresponding to the context. Each policy context has an associated policy context identifier.

In the Policy Decision and Enforcement Subcontract, access decisions are performed by checking permissions that identify resources by name and perhaps action. When a permission is checked, this specification requires identification of the authorization policy context in which the evaluation is to be performed (see Section 4.6, “Setting the Policy Context,” on page 46).

3.1.1.1 Policy Context Life Cycle

Figure 3.1 depicts the policy context life cycle as effected through the methods of the PolicyConfiguration interface. A policy context is in one of three states and all implementations of the PolicyConfiguration interface must implement the state semantics defined in this section.

- open

  A policy context in the open state must be available for configuration by any of the methods of the PolicyConfiguration interface. A policy context in the open state must not be assimilated at Policy.refresh into the policy statements used by the Policy provider in performing its access decisions.

- inService

  A policy context in the inService state must be assimilated at Policy.refresh into the policy statements used by its provider. When a provider's refresh method is called, it must assimilate only policy contexts that are in the inService state and it must ensure that the policy statements put into service for

each policy context are only those defined in the context at the time of the call to refresh. A policy context in the inService state must be unavailable for additional configuration. A policy context in the inService state must be transitioned to the open state when it is returned as a result of a call to getPolicyConfiguration. A policy context is transitioned to the inService state by calling the commit method, and only a policy context in the open state may be transitioned to the inService state.

- deleted

  A policy context in the deleted state must be unavailable for configuration and it must be unavailable for assimilation into its associated Provider. A policy context in the deleted state must be transitioned to the open state when it is returned as a result of a call to getPolicyConfiguration. A policy context is transitioned to the deleted state by calling the delete method.

Note that for a provider implementation to be compatible with multi-threaded environments, it may be necessary to synchronize the refresh method of the provider with the methods of its PolicyConfiguration interface and with the getPolicyConfiguration and inService methods of its PolicyConfigurationFactory.

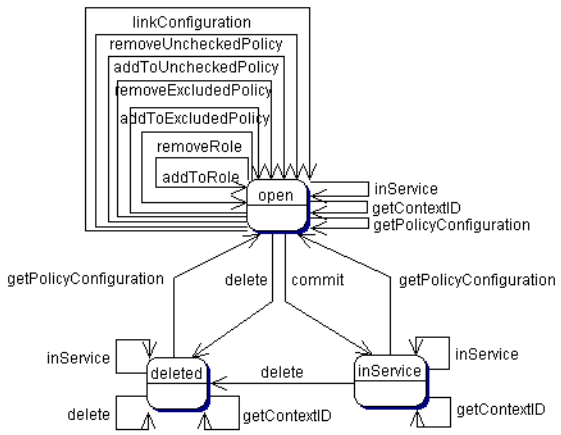


Figure 3.1 PolicyConfiguration State Table

3.1.1.2 Linking Policy Contexts

In the Java EE security model, principal-to-role mappings have application scope; that is, the same principal-to-role mappings must apply in the access decisions applied at all of the modules (that may represent separate policy contexts) that comprise an application. Same application policy contexts must be associated by calling the `PolicyConfiguration.linkConfiguration` method. This method must create a transitive and symmetric relationship within the provider and between this `PolicyConfiguration` and the argument `PolicyConfiguration`, such that they and all PolicyConfiguration objects otherwise linked to either of them share the same principal-to-role mappings. The semantics of the association must preserve the invariant that at most one principal-to-role mapping may apply to any `PolicyConfiguration`.

3.1.2 Servlet Policy Context Identifiers

Servlet requests may be directed to a logical host using various physical or *virtual host* names or addresses, and an application server may be composed of multiple logical hosts. A virtual application server may be realized as a *cluster* of physical application servers, each hosting some subset of the logical hosts of the virtual application server. This specification uses the term *hostname* to refer to the name of a logical host that processes Servlet requests. A servlet container is responsible for mapping the target name or address information of an HTTP request to the appropriate hostname.

To satisfy this specification, an application server must establish servlet policy context identifiers sufficient to differentiate all instances of a web application deployed on the logical host or on any other logical host that may share the same policy statement repository. One way to satisfy this requirement is to compose policy context identifiers by concatenating the hostname with the context path (as defined in the Servlet specification) identifying the web application at the host.

When an application is composed of multiple web modules, a separate policy context must be defined per module. This is necessary to ensure that url-pattern based and servlet name based policy statements configured for one module do not interfere with those configured for another.

3.1.3 Translating Servlet Deployment Descriptors

A reference to a `PolicyConfiguration` object must be obtained by calling the `getPolicyConfiguration` method on the

`PolicyConfigurationFactory` implementation class of the provider configured into the container. The policy context identifier used in the call to the `getPolicyConfiguration` method must be a `String` composed as described in Section 3.1.2, "Servlet Policy Context Identifiers," on page 21. The value `true` must be passed as the second parameter in the call to `getPolicyConfiguration` to ensure that any and all policy statements are removed from the policy context associated with the returned `PolicyConfiguration`. The `security-constraint` and `security-role-ref` elements in the deployment descriptor must be translated into permissions and added to the `PolicyConfiguration` object as defined in the following sections.

3.1.3.1 Translating security-constraint Elements

The paragraphs of this section describe the translation of security-constraints into WebResourcePermission and WebUserDataPermission objects constructed using qualified URL pattern names. In the exceptional case, as defined in "Qualified URL Pattern Names", where a pattern is made irrelevant by a qualifying pattern, the permission instantiations that would result from the translation of the pattern, as described below, must not be performed. Otherwise, the translation of URL patterns in security constraints must yield an equivalent translation to the translation that would result from following the instructions in the remainder of this section.

A WebResourcePermission and a WebUserDataPermission[1] object must be added to the excluded policy statements for each distinct `url-pattern` occurring in the `security-constraint` elements that contain an `auth-constraint` naming no roles (i.e an excluding `auth-constraint`). The permissions must be constructed using the qualified (as defined in "Qualified URL Pattern Names") pattern as their name and with actions defined by the union[2] of the HTTP methods named or implied[3] by all of the collections containing the pattern and occurring in a constraint with an excluding `auth-constraint`. The

---

1. The WebUserDataPermission objects allow a container to determine when to reject a request before redirection if it would ultimately be rejected as the result of an excluding `auth-constraint`.
2. When the union contains the complete the set of all HTTP methods (including all extension methods), the corresponding permission must be constructed with an actions specification that names no HTTP methods (as defined by the chosen permission contructor).
3. A `web-resource-collection` that names no HTTP methods implies that the collection applies to the complete set of HTTP methods (i.e., the methods DELETE, GET, HEAD, OPTIONS, POST, PUT, TRACE, and any method conforming to the "extension-method" syntax defined in IETF RFC 2616 "Hypertext Transfer Protocol -- HTTP/1.1").

constructed permissions must be added to the excluded policy statements by calling the `addToExcludedPolicy` method on the `PolicyConfiguration` object.

A WebResourcePermission must be added to the corresponding role for each distinct combination in the cross-product of `url-pattern` and `role-name` occurring in the `security-constraint` elements that contain an `auth-constraint` naming roles. When an `auth-constraint` names the reserved `role-name`, "*", all of the patterns in the containing `security-constraint` must be combined with all of the roles defined in the web application. Each WebResourcePermission object must be constructed using the qualified pattern as its name and with actions defined by the union of the HTTP methods named or implied by the collections containing the pattern and occurring in a constraint that names (or implies via "*") the role to which the permission is being added. The resulting permissions must be added to the corresponding roles by calling the `addToRole` method on the `PolicyConfiguration` object.

A WebResourcePermission must be added to the unchecked policy statements for each distinct `url-pattern` occurring in the `security-constraint` elements that do not contain an `auth-constraint`. Each WebResourcePermission object must be constructed using the qualified pattern as its name and with actions defined by the union of the HTTP methods named or implied by the collections containing the pattern and occurring in a `security-constraint` without an `auth-constraint`. The resulting permissions must be added to the unchecked policy statements by calling the `addToUncheckedPolicy` method on the `PolicyConfiguration` object.

A WebUserDataPermission must be added to the unchecked policy statements for each distinct combination of `url-pattern` and acceptable connection type resulting from the processing of the `security-constraint` elements that do not contain an excluding `auth-constraint`. The mapping of security-constraint to acceptable connection type must be as defined in "Mapping Transport Guarantee to Connection Type". Each WebUserDataPermission object must be constructed using the qualified pattern as its name and with actions defined by appending[4] a representation of the acceptable connection type to the union of the HTTP methods named or implied by the collections containing the pattern and occurring in a `security-constraint` that maps to the connection type and that does not contain an excluding `auth-constraint`. The resulting

---

4. The value null should be used as the actions value in the construction of a WebUserDataPermission when both the union of the HTTP methods, and the representation of the acceptable connection type may be represented by null. If only one of the action components may be represented by null the other should be used as the actions value.

permissions must be added to the unchecked policy statements by calling the `addToUncheckedPolicy` method on the `PolicyConfiguration` object.

A WebResourcePermission and a WebUserDataPermission must be added to the unchecked policy statements for each `url-pattern` in the deployment descriptor and the default pattern, "/", that is not combined by the `web-resource-collection` elements of the deployment descriptor with every HTTP method value. The permission objects must be constructed using the qualified pattern as their name and with actions represented by an HTTP method exception list that identifies (as defined in "HTTP Method Exception List") all the HTTP methods that do not occur in combination with the pattern.The resulting permissions must be added to the unchecked policy statements by calling the `addToUncheckedPolicy` method on the `PolicyConfiguration` object.

Qualified URL Pattern Names

The URL pattern qualification described in this section serves to capture the best-matching semantics of the Servlet constraint model in the qualified names such that the WebResourcePermission and WebUserDataPermission objects can be tested using the standard Java SE permission evaluation logic.

The WebResourcePermission and WebUserDataPermission objects resulting from the translation of a Servlet deployment descriptor must be constructed with a name produced by qualifying the URL pattern. The rules for qualifying a URL pattern are dependent on the rules for determining if one URL pattern matches another as defined in Section 3.1.3.3, "Servlet URL-Pattern Matching Rules", and are described as follows:

- If the pattern is a path prefix pattern, it must be qualified by every path-prefix pattern in the deployment descriptor matched by and different from the pattern being qualified. The pattern must also be qualified by every exact pattern appearing in the deployment descriptor that is matched by the pattern being qualified.
- If the pattern is an extension pattern, it must be qualified by every path-prefix pattern appearing in the deployment descriptor and every exact pattern in the deployment descriptor that is matched by the pattern being qualified.
- If the pattern is the default pattern, "/", it must be qualified by every other pattern except the default pattern appearing in the deployment descriptor.
- If the pattern is an exact pattern, its qualified form must not contain any qualifying patterns.

URL patterns are qualified by appending to their String representation, a colon separated representation of the list of patterns that qualify the pattern. Duplicates must not be included in the list of qualifying patterns, and any qualifying pattern matched by another qualifying pattern may[5] be dropped from the list.

```
QualifyingPatternList ::=
    empty string | colon QualifyingPattern |
    QualifyingPatternList colon QualifyingPattern

QualifiedPattern ::= Pattern QualifyingPatternList
```

Any pattern, qualified by a pattern that matches it, is overridden and made irrelevant (in the translation) by the qualifying pattern. Specifically, all extension patterns and the default pattern are made irrelevant by the presence of the path prefix pattern "/*" in a deployment descriptor. Patterns qualified by the "/*" pattern violate the URLPatternSpec constraints of WebResourcePermission and WebUserDataPermission names and must be rejected by the corresponding permission constructors.

HTTP Method Exception List

An HTTP method exception list is used to represent, by set difference, a non-enumerable subset of the set of all possible HTTP methods. An exception list respresents the subset of the complete set of HTTP methods formed by subtracting the methods named in the exception list from the complete set.

An exception list is distinguished by its first character, which must be the exclaimation point (i.e., “!”) character. A comma seperated list of one or more HTTP method names must follow the exclaimation point. The syntax of an HTTP method list is formally defined as follows:

```
ExtensionMethod ::= any token as defined by IETF RFC 2616
    (i.e., 1*[any CHAR except CTLS or separators as defined in RFC 2616])

HTTPMethod ::= “GET” | “POST” | “PUT” | “DELETE” | “HEAD” |
    “OPTIONS” | “TRACE” | ExtensionMethod

HTTPMethodList ::= HTTPMethod | HTTPMethodList comma HTTPMethod

HTTPMethodExceptionList ::= exclaimationPoint HTTPMethodList
```

5. Qualifying patterns implied by another qualifying pattern may be dropped because the use of the reduced list to qualify a pattern will yield a URLPatternSpec “equal” to the URL-PatternSpec produced by qualifying the pattern with the full list (for example, /a/*:/a/b:/a/b/*:/a/b/c/* is equal to /a/*:/a/b/*).

Mapping Transport Guarantee to Connection Type

A `transport-guarantee` (in a user-data-constraint) of NONE, or a `security-constraint` without a `user-data-constraint`, indicates that the associated URL patterns and HTTP methods may be accessed over any (including an unprotected) transport. A `transport-guarantee` of INTEGRAL indicates that acceptable connections are those deemed by the container to be integrity protected. A `transport-guarantee` of CONFIDENTIAL indicates that acceptable connections are those deemed by the container to be protected for confidentiality.

TABLE 3-1 transport-guarantee to Acceptable Connection Mapping

transport-guarantee in constraint	*connection type String value*
INTEGRAL	":INTEGRAL"
CONFIDENTIAL	":CONFIDENTIAL"
NONE (including no user-data-constraint)	null

3.1.3.2 Translating Servlet security-role-ref Elements

For each `security-role-ref` appearing in the deployment descriptor a corresponding WebRoleRefPermission must be added to the corresponding role. The name of the WebRoleRefPermission must be the `servlet-name` in whose context the `security-role-ref` is defined. The actions of the WebRoleRefPermission must be the value of the `role-name` (that is the reference), appearing in the `security-role-ref`. The deployment tools must call the `addToRole` method on the `PolicyConfiguration` object to add the WebRoleRefPermission object resulting from the translation to the `role` identified in the `role-link` appearing in the `security-role-ref`.

Additional WebRoleRefPermission objects must be added to the `PolicyConfiguration` as follows. For each servlet element in the deployment descriptor a WebRoleRefPermission must be added to each `security-role` of the application whose name does not appear as the `role-name` in a `security-role-ref` within the servlet element. The name of each such WebRoleRefPermission must be the `servlet-name` of the corresponding servlet element. The actions (that is, reference) of each such WebRoleRefPermission must be the corresponding (non-appearing) `role-name`. The resulting permissions must be added to the corresponding roles by calling the `addToRole` method on the `PolicyConfiguration` object.

For each `security-role` defined in the deployment descriptor, an additional WebRoleRefPermission must[6] be added to the corresponding role by calling the `addToRole` method on the `PolicyConfiguration` object. The name of all such permissions must be the empty string, and the actions of each such permission must be the `role-name` of the corresponding role.

3.1.3.3 Servlet URL-Pattern Matching Rules

This URL pattern matches another pattern if they are related, by case sensitive comparison, as follows:

- their pattern values are String equivalent, or
- this pattern is the path-prefix pattern "/*", or
- this pattern is a path-prefix pattern (that is, it starts with "/" and ends with "/*") and the other pattern starts with the substring of this pattern, minus its last 2 characters, and the next character of the other pattern, if there is one, is "/", or
- this pattern is an extension pattern (that is, it starts with "*.") and the other pattern ends with this pattern, or
- this pattern is the special default pattern, "/", which matches all other patterns.

TABLE 3-2 url-pattern Types by Example

pattern type	example
exact	/acme/widget/hammer
path prefix	/acme/widget/*
extension	*.html
default	/

3.1.3.4 Example

This example demonstrates the WebResourcePermission and WebUserDataPermission objects that would result from the translation of a deployment descriptor that contained the following `security-constraint` elements.

---

[6] These additional WebRoleRefPermission objects support the use of isUserInRole from unmapped (to a Servlet) JSP components.

```
<!--
The following security-constraint excludes access to the patterns
and method combinations defined by the two contained web-re-
source-collections. The first collection explicitly names the
HTTP methods DELETE and PUT, while the second collection implic-
itly applies to all the HTTP methods.
-->
<security-constraint>
   <web-resource-collection>
      <web-resource-name>sc1.c1</web-resource-name>
      <url-pattern>/a/*</url-pattern>
      <url-pattern>/b/*</url-pattern>
      <url-pattern>/a</url-pattern>
      <url-pattern>/b</url-pattern>
      <http-method>DELETE</http-method>
      <http-method>PUT</http-method>
   </web-resource-collection>

   <web-resource-collection>
      <web-resource-name>sc1.c2</web-resource-name>
      <url-pattern>*.asp</url-pattern>
   </web-resource-collection>
   <auth-constraint/>
</security-constraint>

<!--
The following security-constraint restricts access to the pat-
terns and method combinations defined by the two contained web-
resource-collections to callers in role R1 who connect using a
confidential transport.
-->
<security-constraint>
   <web-resource-collection>
      <web-resource-name>sc2.c1</web-resource-name>
      <url-pattern>/a/*</url-pattern>
      <url-pattern>/b/*</url-pattern>
      <http-method>GET</http-method>
   </web-resource-collection>

   <web-resource-collection>
      <web-resource-name>sc2.c2</web-resource-name>
      <url-pattern>/b/*</url-pattern>
      <http-method>POST</http-method>
   </web-resource-collection>
```

```
    <auth-constraint>
        <role-name>R1</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

TABLE 3-3 contains the qualified URL pattern names that would result from the translation of the `security-constraint` elements (including the qualified form of the default pattern). The second column of TABLE 3-3 contains the canonical form of the qualified names. The values in the second column have been derived from the values in the first column by removing qualifying patterns matched by other qualifying patterns.

TABLE 3-3 Qualified URL Pattern Names from Example

Qualified URL Pattern Name	*Canonical Form*
/a	/a
/b	/b
/a/*:/a	/a/*:/a
/b/*:/b	/b/*:/b
.asp:/a/:/b/*	*.asp:/a/*:/b/*
/:/a:/b:/a/*:/b/*:*.asp	/:/a/*:/b/*:*.asp

TABLE 3-4 represents the permissions and PolicyConfiguration operations that would result from the translation of the security-constraint elements.The names appearing in the second column of the table are those found in the first column of TABLE 3-3. As noted previously, any equivalent form of the qualified names, including their canonical forms, could have been used in the permission constructions.

TABLE 3-4 Permissions and PolicyConfiguration Operations from Example

Permission Type	*Name*	*Actions*	*Policy Configuration Add To*
WebResource	/a/*:/a	DELETE,PUT	excluded
WebUserData	/a/*:/a	DELETE,PUT	excluded
WebResource	/b/*:/b	DELETE,PUT	excluded
WebUserData	/b/*:/b	DELETE,PUT	excluded
WebResource	/a	DELETE,PUT	excluded
WebUserData	/a	DELETE,PUT	excluded
WebResource	/b	DELETE,PUT	excluded
WebUserData	/b	DELETE,PUT	excluded
WebResource	*.asp:/a/*:/b/*	null[1]	excluded
WebUserData	*.asp:/a/*:/b/*	null	excluded
WebResource	/a/*:/a	GET	role(R1)
WebResource	/b/*:/b	GET,POST	role(R1)
WebUserData	/a/*:/a	GET:CONFIDENTIAL	unchecked
WebUserData	/b/*:/b	GET,POST:CONFIDENTIAL	unchecked
WebResource	/a/*:/a	!DELETE,GET,PUT	unchecked
WebUserData	/a/*:/a	!DELETE,GET,PUT	unchecked
WebResource	/b/*:/b	!DELETE,GET,POST,PUT	unchecked
WebUserData	/b/*:/b	!DELETE,GET,POST,PUT	unchecked
WebResource	/a	!DELETE,PUT	unchecked
WebUserData	/a	!DELETE,PUT	unchecked
WebResource	/b	!DELETE,PUT	unchecked

TABLE 3-4 Permissions and PolicyConfiguration Operations from Example

Permission Type	*Name*	*Actions*	*Policy Configuration Add To*
WebUserData	/b	!DELETE,PUT	unchecked
WebResource	/:/a:/b:/a/*:/b/*:*.asp	null	unchecked
WebUserData	/:/a:/b:/a/*:/b/*:*.asp	null	unchecked

1.The canonical form for the set of all HTTP Methods (including all extension methods) is null.

3.1.4 EJB Policy Context Identifiers

To satisfy this specification, an application server must establish EJB policy context identifiers sufficient to differentiate all instances of the deployment of an EJB jar on the application server, or on any other application server with which the server may share the same policy statement repository.

When an application is composed of multiple EJB jars, no two jars that share at least one ejb-name value in common may share the same policy context identifiers.

3.1.5 Translating EJB Deployment Descriptors

A reference to a `PolicyConfiguration` object must be obtained by calling the `getPolicyConfiguration` method on the `PolicyConfigurationFactory` implementation class of the provider configured into the container. The policy context identifier used in the call to `getPolicyConfiguration` must be a `String` that satisfies the requirements described in Section 3.1.4, “EJB Policy Context Identifiers,” on page 31. The value `true` must be passed as the second parameter in the call to `getPolicyConfiguration` to ensure that any and all policy statements are removed from the policy context associated with the returned `PolicyConfiguration`. The `method-permission, exclude-list,` and `security-role-ref` elements appearing in the deployment descriptor must be translated into permissions and added to the `PolicyConfiguration` object to yield an equivalent translation as that defined in the following sections and such that every EJB method for which the container performs pre-dispatch access decisions is implied by at least one permission resulting from the translation.

3.1.5.1 Translating EJB method-permission Elements

For each `method` element of each `method-permission` element, an EJBMethodPermission object translated from the `method` element must be added to the policy statements of the `PolicyConfiguration` object. The name of each such EJBMethodPermission object must be the `ejb-name` from the corresponding `method` element, and the actions must be established by translating the `method` element into a method specification according to the methodSpec syntax defined in the documentation of the EJBMethodPermission class. The actions translation must preserve the degree of specificity with respect to `method-name`, `method-intf`, and `method-params` inherent in the `method` element.

If the `method-permission` element contains the `unchecked` element, then the deployment tools must call the `addToUncheckedPolicy` method to add the permissions resulting from the translation to the `PolicyConfiguration` object. Alternatively, if the `method-permission` element contains one or more `role-name` elements, then the deployment tools must call the `addToRole` method to add the permissions resulting from the translation to the corresponding roles of the `PolicyConfiguration` object.

3.1.5.2 Translating the EJB exclude-list

An EJBMethodPermission object must be created for each `method` element occurring in the `exclude-list` element of the deployment descriptor. The name and actions of each EJBMethodPermission must be established as described in Section 3.1.5.1, "Translating EJB method-permission Elements."

The deployment tools must use the `addToExcludedPolicy` method to add the EJBMethodPermission objects resulting from the translation of the `exclude-list` to the excluded policy statements of the `PolicyConfiguration` object.

3.1.5.3 Translating EJB security-role-ref Elements

For each `security-role-ref` element appearing in the deployment descriptor, a corresponding EJBRoleRefPermission must be created. The name of each EJBRoleRefPermission must be obtained as described for EJBMethodPermission objects. The actions used to construct the permission must be the value of the `role-name` (that is the reference), appearing in the `security-role-ref`. The deployment tools must call the `addToRole` method on the `PolicyConfiguration` object to add a policy statement

corresponding to the EJBRoleRefPermission to the role identified in the `role-link` appearing in the `security-role-ref`.

3.1.6 Deploying an Application or Module

The application server's deployment tools must translate the declarative authorization policy appearing in the application or module deployment descriptor(s) into policy statements within the Policy providers used by the containers to which the components of the application or module are being deployed.

When a module is deployed, its policy context must be linked to all the other policy contexts with which it must share the same principal-to-role mapping. When an application is deployed, every policy context of the application must be linked to every other policy context of the application with which it shares a common Policy provider. Policy contexts are linked[7] by calling the linkConfiguration method on the PolicyConfiguration objects of the provider.

After the `PolicyConfiguration` objects are linked, the `commit` method must be called on all the `PolicyConfiguration` objects to place them in service such that their policy statements will be assimilated by the corresponding Policy providers.

Once the translation, linking, and committing has occurred, a call must be made to `Policy.refresh` on the Policy provider used by each of the containers to which the application or module is being deployed. The calls to `Policy.refresh` must occur before the containers will accept requests for the deployed resources.

The policy context identifiers corresponding to the deployed application or module must be recorded in the application server so that they can be used by containers to establish the policy context as required by Section 4.6, "Setting the Policy Context" of the Policy Decision and Enforcement Subcontract, and such that the Deployer may subsequently remove or modify the corresponding policy contexts as a result of the undeployment or redeployment of the application.

---

7. Policy context linking is transitive and symmetric, and this specification should not be interpreted as requiring that linkConfiguration be called on every combination of policy contexts that must share the same principal-to-role mapping.

3.1.7 Undeploying an Application or Module

To ensure that there is not a period during undeployment when the removal of policy statements on application components renders what were protected components unprotected, the application server must stop dispatching requests for the application's components before undeploying an application or module.

To undeploy an application or module, the deployment tools must indicate at all the Policy providers to which policy contexts of the application or module have been deployed that the policy contexts associated with the application or module that have been configured in the provider are to be removed from service. A deployment tool indicates that a policy context is to be removed from service either by calling getPolicyConfiguration with the identifier of the policy context on the provider's PolicyConfigurationFactory or by calling delete on the corresponding PolicyConfiguration object. If the getPolicyConfiguration method is used, the value `true` should be passed as the second argument to cause the corresponding policy statements to be deleted from the context. After the policy contexts are marked for removal from service, a call must be made to `Policy.refresh` on all of the Policy providers from which at least one module of the application or module was marked for removal from service.

3.1.8 Deploying to an existing Policy Configuration

Containers are not required to deploy to an existing policy configuration. Containers that chose to provide this functionality must satisfy the following requirements.

To associate an application or module with an existing set of linked policy contexts, the identifiers of the existing policy contexts must be applied by the relevant containers in fulfilling their obligations as defined in the Policy Decision and Enforcement Subcontract. The policy contexts should be verified for existence, by calling the `inService` method of the `PolicyConfigurationFactory` of the Policy providers of the relevant containers. The deployment tools must call `Policy.refresh` on the Policy provider of each of the relevant containers, and the containers must not perform pre-dispatch decisions or dispatch requests for the deployed resources until these calls have completed.

3.1.9 Redeploying a Module

Containers are not required to implement redeployment functionality. Containers that chose to provide this functionality must satisfy the following requirements.

To ensure redeployment does not create a situation where the removal of policy statements on application components renders what were protected components unprotected, the application server must stop dispatching requests for the application's components before redeployment begins. The application server must not resume dispatching requests for the application's components until after the calls to Policy.refresh, described below, have completed.

To redeploy a module, the deployment tools must indicate at all of the Policy providers to which the module is to be redeployed that the policy context associated with the module is to be removed from service. If the module is to be redeployed to the same policy context at a provider, all policy statements must be removed from the policy context at the provider. After the policy contexts have been marked for removal from service and emptied of policy statements (as necessary), the deployment tools must translate the declarative authorization policy appearing in the module's deployment descriptor into PolicyConfiguration objects within the Policy providers at which the module is being redeployed. After the translation, the `PolicyConfiguration` objects must be linked, as necessary, to every other policy context in the same provider with which they must share the same principal-to-role mappings. After the `PolicyConfiguration` objects are linked, the `commit` method must be called on the `PolicyConfiguration` objects. After commit has been called, a call must be made to `Policy.refresh` on the Policy provider used by each of the containers to which the module has been redeployed.

3.2 What the Provider Must Do

The provider must include an implementation of the `javax.security.jacc.PolicyConfigurationFactory` class along with a matched implementation of a class that implements the `javax.security.jacc.PolicyConfiguration` interface. In addition to providing a `PolicyConfiguration` interface for integration with the application server's deployment tools, the provider must also include a management interface for policy administrators to use to grant the collections of permissions that comprise roles, to principals. This interface need not be standardized.

The provider must ensure that all of the permissions added to a role in a policy context are granted to any principal mapped to the role by the policy administrator. The provider must ensure that the same principal-to-role mappings are applied to all linked policy contexts.

The provider must ensure that excluded policy statements take precedence over overlapping unchecked policy statements, and that both excluded and unchecked policy statements take precedence over overlapping role based policy statements.

This specification does not prescribe the policy language or the methods used within providers to implement the policy and role requirements described above.

3.3 Permission to Configure Policy

The `getPolicyConfigurationFactory`, and `inService` methods of the abstract factory class, `javax.security.jacc.PolicyConfigurationFactory`, must throw a SecurityException when called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

The `getPolicyConfiguration` method of all implementations of the `PolicyConfigurationFactory` abstract class must throw a SecurityException when called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

All of the public methods of all of the concrete implementations of the `PolicyConfiguration` interface must throw a SecurityException when called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

In cases where a required permission is not held by a caller, the implementation must return without changing the state of the policy statement repository.

The containers of an application server must be granted the "getPolicy" SecurityPermission and the "setPolicy" SecurityPermission. J2EE 1.3 Containers that choose to support this contract must be granted the "getPolicy" AuthPermission and the "setPolicy" AuthPermission.

CHAPTER 4

Policy Decision and Enforcement Subcontract

The Policy Decision and Enforcement Subcontract defines the interactions between container policy enforcement points and the providers that implement the policy decisions required by Java EE containers.

4.1 Policy Enforcement by Servlet Containers

Servlet containers must employ the methods defined in the following subsections to enforce the authorization policies established for web resources.

4.1.1 Permission Names for Transport and Pre-Dispatch Decisions

The name of the permission checked in a transport or pre-dispatch decision must be the unqualified request URI minus the context path. For the special case where this transformation of the request URI yields the URLPattern "/", the empty string URLPattern, "", must be used as the permission name.

For the special case where the empty string must be substituted for the "/" pattern in the permission evaluation, all target related processing (including servlet mapping, filter mapping, and form based login processing) must be performed using the original pattern, "/".

4.1.2 Evaluation of Transport Guarantees

The Servlet container must obtain a WebUserDataPermission object with name obtained from the request URI as defined in Section 4.1.1, “Permission Names for Transport and Pre-Dispatch Decisions”. The actions of the obtained permission

must be composed of the HTTP method of the request and a protection value describing the transport layer protection of the connection on which the request arrived. The protection value must be as follows:

- If the request arrived on a connection deemed by the container to be protected for confidentiality, a protection value of ":CONFIDENTIAL" must be used.
- If the request arrived on a connection deemed by the container to be protected for integrity (but not confidentiality), a protection value of ":INTEGRAL" must be used.
- If the request arrived on a connection deemed by the container to be unprotected, the actions used in the permission construction must contain only the HTTP method of the request.

The Servlet container must use one of the methods described in Section 4.7, "Checking AccessControlContext Independent Grants" to test if access to the resource using the method and connection type encapsulated in the WebUserDataPermission is permitted. If a SecurityException is thrown in the permission determination, it must be caught, and the result of the determination must be that access to the resource using the method and connection type is not permitted. If access is not permitted, the request must be redirected as defined by the Servlet Specification. If access is permitted, the request must be subjected to a pre-dispatch decision.

4.1.3 Pre-dispatch Decision

The Servlet container must obtain a WebResourcePermission object with name obtained from the request URI as defined in Section 4.1.1, "Permission Names for Transport and Pre-Dispatch Decisions". The actions of the obtained permission must be the HTTP method of the request. The Servlet container must use one of the methods described in Section 4.8, "Checking the Caller for a Permission" to test if the WebResourcePermission has been granted to the caller. If a SecurityException is thrown in the permission determination, it must be caught, and the result of the determination must be that the permission is not granted to the caller. The Servlet container may only dispatch the request to the web resource if the WebResourcePermission is determined to be granted to the caller. Otherwise the request must be rejected with the appropriate HTTP error message as defined by the Servlet Specification.

Before it dispatches a call to a web resource, the container must associate with the call thread an AccessControlContext containing the principals of (only) the

target component's runAs identity (as defined in Section 4.5, "Component runAs Identity).

4.1.4 Application Embedded Privilege Test

When a call is made from a web resource to `isUserInRole(String roleName)` the implementation of this method must obtain a WebRoleRefPermission object with name corresponding to the servlet-name of the calling web resource and with actions equal to the roleName used in the call. For the special case where the call to `isUserInRole` is made from a web resource that is not mapped to a Servlet (i.e. by a `servlet-mapping`), the name of the WebRoleRefPermission must be the empty string. In either case, the implementation of the isUserInRole method must then use one of the methods described in Section 4.8, "Checking the Caller for a Permission" to determine if the WebRoleRefPermission has been granted to the caller. If a SecurityException is thrown in the permission determination, it must be caught, and the result of the determination must be that the permission is not granted to the caller. If it is determined that the WebRoleRefPermission has been granted to the caller, isUserInRole must return true. Otherwise the return value must be false.

4.2 Provider Support for Servlet Policy Enforcement

In support of the policy enforcement done by servlet containers, providers must implement the policy decision functionality defined in the following subsections.

4.2.1 Servlet Policy Decision Semantics

A Policy provider must use the combined policy statements of the default policy context (as defined in Section 4.10, "Default Policy Context") and of the policy context identified by calling PolicyContext.getContextID to determine if they imply the permission being checked. If one or more excluded policy statements imply the checked permission, the evaluation may terminate and the checked permission must be determined not to be granted. Otherwise, if one or more unchecked policy statements imply the checked permission, the checked permission must be determined to be granted independent of AccessControlContext. If the status of the checked permission is not resolved by the excluded and unchecked evaluations, it must be determined if a permission that implies the checked permission has been granted to the AccessControlContext being tested for the permission. The checked permission

may only be determined to be granted if a permission that implies the checked permission has been granted to the AccessControlContext. Otherwise the permission must be determined not to be granted. The policy decision semantics are dependent on permission specific rules for determining if the permissions in policy statements imply the permission being checked.

The WebResourcePermission, WebUserDataPermission, and WebRoleRefPermission specific rules used to determine if the permissions in policy statements imply a checked permission are defined in the next sections.

4.2.1.1 Matching Qualified URL Pattern Names

Qualified URL Patterns names were described in a subsection of Section 3.1.3.1, "Translating security-constraint Elements". The WebResourcePermission and WebUserDataPermission classes use the term URLPatternSpec to describe the syntax of qualified URL pattern names. The URLPatternSpec syntax is defined as follows:

```
URLPatternList ::= URLPattern | URLPatternList colon URLPattern
URLPatternSpec ::= URLPattern | URLPattern colon URLPatternList
name ::= URLPatternSpec
```

Given this syntax, A reference URLPatternSpec matches an argument URLPatternSpec if all of the following are true.

- The first URLPattern in the argument URLPatternSpec is matched by the first URLPattern in the reference URLPatternSpec.
- The first URLPattern in the argument URLPatternSpec is NOT matched by any URLPattern in the URLPatternList of the reference URLPatternSpec.
- If the first URLPattern in the argument URLPatternSpec matches the first URLPattern in the reference URLPatternSpec, then every URLPattern in the URLPatternList of the reference URLPatternSpec must be matched by a URLPattern in the URLPatternList of the argument URLPatternSpec.

The comparisons described above are case sensitive, and all matching is according to the rules defined in Section 3.1.3.3, "Servlet URL-Pattern Matching Rules".

4.2.1.2 Matching HTTP Method Specifications

The WebResourcePermission and WebUserDataPermission classes use the term HTTPMethodSpec to describe the syntax of the HTTP method component of their actions values. The HTTPMethodSpec syntax is defined as follows:

```
HTTPMethodSpec ::= null | emptyString |
   HTTPMethodExceptionList | HTTPMethodList
```

Given this syntax, a reference HTTPMethodSpec matches an argument HTTPMethodSpec if all of the HTTP methods represented by the actions of the argument specification are included in the method subset represented by the actions of the reference specification.

A null or emptyString HTTPMethodSpec represents the entire set of HTTP methods, and as such, matches any argument HTTPMethodSpec. An HTTPMethodExceptionList[1] matches any subset that does not include a method named in the exception list. A reference HTTPMethodList matches an argument HTTPMethodList if the methods named in the argument list are all named in the reference list. An HTTPMethodList never matches an argument HTTPMethodExceptionList. Neither an HTTPMethodList or an HTTPMethodExceptionList matches a null or emptyString HTTPMethodSpec.

4.2.1.3 WebResourcePermission Matching Rules

A reference WebResourcePermission implies an argument permission if all of the following are true.

- The argument permission is an instanceof WebResourcePermission.
- The name of the argument permission is matched by the name of the reference permission according to the rules defined in Section 4.2.1.1, “Matching Qualified URL Pattern Names.
- The HTTP methods represented by the actions of the argument permission are a subset of the HTTP methods represented by the actions of the reference permission as defined in Section 4.2.1.2, “Matching HTTP Method Specifications”.

The comparisons described above are case sensitive.

4.2.1.4 WebRoleRefPermission Matching Rules

A reference WebRoleRefPermission implies an argument permission if all of the following are true.

---

1. The syntax and semantics of an HTTPMethodExceptionList are described in a subsection of Section 3.1.3.1, “Translating security-constraint Elements”.

- The argument permission is an instanceof WebRoleRefPermission.
- The name of the argument permission is equivalent to the name of the reference permission.
- The actions (i.e role reference) of the argument permission is equivalent to the actions (i.e role reference) of the reference permission.

  The comparisons described above are case sensitive.

4.2.1.5 WebUserDataPermission Matching Rules

A reference WebUserDataPermission implies an argument permission if all of the following are true.

- The argument permission is an instanceof WebUserDataPermission.
- The name of the argument permission is matched by the name of the reference permission according to the rules defined in Section 4.2.1.1, “Matching Qualified URL Pattern Names.
- The HTTP methods represented by the actions of the argument permission are a subset of the HTTP methods represented by the actions of the reference permission as defined in Section 4.2.1.2, “Matching HTTP Method Specifications”.
- The transportType in the actions of the reference permission either corresponds to the value "NONE", or equals the transportType in the actions of the argument permission.

  The comparisons described above are case sensitive.

4.3 Policy Enforcement by EJB Containers

EJB containers must employ the methods defined in the following subsections to enforce the authorization policies established for EJB resources.

4.3.1 EJB Pre-dispatch Decision

The EJB container must obtain an EJBMethodPermission object with name corresponding to the ejb-name of the target resource and with actions that completely specify the about-to-be-called method of the EJB by identifying the

method interface, method name, and method signature as defined for a methodSpec in the documentation of the EJBMethodPermission class.

The EJB container must use one of the methods described in Section 4.8, "Checking the Caller for a Permission" to determine if the EJBMethodPermission has been granted to the caller. If a SecurityException is thrown in the permission determination, it must be caught, and the result of the determination must be that the permission is not granted to the caller. The EJB container may only dispatch the request to the EJB resource, if the EJBMethodPermission is determined to be granted to the caller. Otherwise the request must be rejected with the appropriate RMISecurityException, as defined by the EJB specification.

Before it dispatches a call to an EJB, the container must associate with the call thread an AccessControlContext containing the principals of only the target EJB's runAs identity (as defined in Section 4.5, "Component runAs Identity).

4.3.2 EJB Application Embedded Privilege Test

When an EJB makes a call to `isCallerInRole(String roleName)` the implementation of this method must obtain an EJBRoleRefPermission object with name corresponding to the `ejb-name` of the EJB making the call and with actions equal to the `roleName` used in the call. The implementation of the isCallerInRole method must then use one of the methods described in Section 4.8, "Checking the Caller for a Permission" to determine if the EJBRoleRefPermission has been granted to the caller. If a SecurityException is thrown in the permission determination, it must be caught, and the result of the determination must be that the permission is not granted to the caller. If it is determined that the EJBRoleRefPermission has been granted to the caller, then isCallerInRole must return true. Otherwise the return value must be false.

4.4 Provider Support for EJB Policy Enforcement

In support of the policy enforcement done by EJB containers, providers must implement the policy decision functionality defined in the following subsections.

4.4.1 EJB Policy Decision Semantics

A Policy provider must employ the policy decision semantics described in Section 4.2.1, "Servlet Policy Decision Semantics" in the Processing of EJB Policy decisions.

The EJBMethodPermission and EJBRoleRefPermission specific rules used to determine if the permissions in policy statements imply a checked permission are defined in the following sections.

4.4.1.1 EJBMethodPermission Matching Rules

A reference EJBMethodPermission implies an argument permission, if all of the following are true.

- The argument permission is an instanceof EJBMethodPermission.
- The name of the argument permission is equivalent to the name of the reference permission.
- The methods to which the argument permission applies (as defined in its actions) must be a subset of the methods to which the reference permission applies (as defined in its actions). This rule is satisfied if all of the following conditions are met.

    - The method name of the reference permission is null, the empty string, or equivalent to the method name of the argument permission.
    - The method interface of the reference permission is null, the empty string, or equivalent to the method interface of the argument permission.
    - The method parameter type list of the reference permission is null, the empty string, or equivalent to the method parameter type list of the argument permission.

  The comparisons described above are case sensitive.

TABLE 4-1 demonstrate the properties of EJBMethodPermission matching by example.

TABLE 4-1 EJBMethodPermission methodSpec Matching Examples

type	*methodInterface Spec*	*methodName Spec*	*methodParams Spec*	*implies checked permission*
checked permission	Home	doThis	java.lang.String	
reference permission	empty string	empty string	empty string	yes
reference permission	Home	empty string	empty string	yes
reference permission	empty string	doThis	empty string	yes
reference permission	empty string	empty string	java.lang.String	yes
reference permission	Remote	doThis	java.lang.String	no
reference permission	Home	doNotDoThis	java.lang.String	no
reference permission	Home	doThis	java.lang.byte	no

4.4.1.2 EJBRoleRefPermission Matching Rules

A reference EJBRoleRefPermission implies an argument permission, if all of the following are true.

- The argument permission is an instanceof EJBRoleRefPermission.
- The name of the argument permission is equivalent to the name of the reference permission.
- The actions (i.e role reference) of the argument permission is equivalent to the actions (i.e role reference) of the reference permission.

  The comparisons described above are case sensitive.

4.5 Component runAs Identity

The identity used by Servlet or EJB components in the operations they perform is configured by the Deployer. This identity is referred to as the component's runAs identity. By default (and unless otherwise specified in the EJB or Servlet

specifications), components are configured such that they are assigned the identity of their caller (such as it is) as their runAs identity. Alternatively, a Deployer may choose to assign an environment specific identity as a component's runAs identity. In this case, the container must establish the specified identity as the component's runAs identity independent of the identity of the component's caller.

When a Deployer configures an environment specific component identity based on a deployment descriptor specification that the component run with an identity mapped to a role, those responsible for defining the principal-to-role mapping must ensure that the specified identity is mapped to the role.

A container establishes a component's runAs identity by associating an AccessControlContext with the component's thread of execution. The container must ensure that the AccessControlContext includes a SubjectDomainCombiner; and the container must protect the AccessControlContext associated with a running component such that, by default, the component is not granted permissions sufficient to modify the AccessControlContext.

4.6 Setting the Policy Context

A policy context identifier is set on a thread by calling the setContextID method on the PolicyContext utility class. The value of a thread's policy context identifier is *null* until the setContextID method is called. Before invoking Policy to evaluate a transport guarantee or to perform a pre-dispatch decision, and before dispatching into a Servlet or EJB component, a container must ensure that the thread's policy context identifier identifies the policy context corresponding to the instance of the module or application for which the operation is being performed.

Containers must be granted the "setPolicy" SecurityPermission independent of policy context identifier (or in all policy contexts) as they need this permission to set the policy context identifier.

4.6.1 Policy Context Handlers

This specification requires that containers register policy context handlers with the PolicyContext utility class such that Policy providers can invoke these handlers to obtain additional context to apply in their access decisions. Policy context handlers are objects that implement the PolicyContextHandler interface. To satisfy the requirements of this specification, containers are required to provide and register with the PolicyContext class the policy context handlers described in the following subsections. All of the required context handlers must return the value null when activated outside of the scope of a container's processing of a

component request. In this context, the scope of a container's processing of a component request begins when the container asks policy to perform the corresponding pre-dispatch access decision and ends either when the access decision returns a failed authorization or when the dispatched request returns from the component to the container.

Policy providers must not call methods on or modify the objects returned by the context handlers if these actions will cause the container to fail in its processing of the associated request.

Containers may delay the registration of the required context handlers until the first call to `PolicyContext.getHandlerKeys`, or for a specific handler, until the required context handler is activated (assuming `getHandlerKeys` has not been called).

4.6.1.1 Container Subject Policy Context Handler

All EJB and Servlet containers must register a PolicyContextHandler whose getContext method returns a javax.security.auth.Subject object when invoked with the key "javax.security.auth.Subject.container". When this handler is activated as the result of a policy decision performed by a container before dispatch into a component, this handler must return a Subject containing the principals and credentials of the "caller" of the component. When activated from the scope of a dispatched call, this handler must return a Subject containing the principals and credentials corresponding to the identity established by the container prior to the activation of the handler. The identity established by the container will either be the component's runAs identity or the caller's identity (e.g. when an EJB component calls `isCallerInRole`). In all cases, if the identity of the corresponding Subject has not been established or authenticated, this handler must return the value null.

4.6.1.2 SOAPMessage Policy Context Handler

All EJB containers must register a PolicyContextHandler whose getContext method returns a javax.xml.soap.SOAPMessage object when invoked with the key "javax.xml.soap.SOAPMessage". If the request being processed by the container arrived as a SOAP request at the ServiceEndpoint method interface, the container must return the SOAP message object when this handler is activated. Otherwise, this handler must return the value null.

4.6.1.3 HttpServletRequest Policy Context Handler

All Servlet containers must register a PolicyContextHandler whose getContext method returns a javax.servlet.http.HttpServletRequest object when invoked with the key "javax.servlet.http.HttpServletRequest". When this handler is activated, the container must return the HttpServletRequest object corresponding to the component request being processed by the container.

4.6.1.4 EnterpriseBean Policy Context Handler

All EJB containers must register a PolicyContextHandler whose getContext method returns a javax.ejb.EnterpriseBean object when invoked with the key "javax.ejb.EnterpriseBean". When this handler is activated, the container must return the EnterpriseBean object corresponding to the EJB component request (as restricted below) being processed by the container. The EnterpriseBean object must only be returned when this handler is activated within the scope of a container's processing of a business method of the EJB Remote, Local, or ServiceEndpoint interfaces of the EnterpriseBean object. The value null must be returned if the bean implementation class does not implement the javax.ejb.EnterpriseBean interface.

4.6.1.5 EJB Arguments Policy Context Handler

All EJB containers must register a PolicyContextHandler whose getContext method returns an array of objects (Object[]) containing the arguments of the EJB method invocation (in the same order as they appear in the method signature) when invoked with the key "javax.ejb.arguments". The context handler must return the value null when called in the context of a SOAP request that arrived at the ServiceEndpoint method interface. Otherwise, the context handler must return the array of objects corresponding to the parameters of the EJB component invocation. If there are no parameters in the method signature, the context handler must return an empty array of Object (i.e. Object[0]).

4.7 Checking AccessControlContext Independent Grants

This section describes the techniques used by containers to check permissions for which policy is defined in terms of the operation defined by the permission and independent of properties of the invocation context represented in the AccessControlContext. The WebUserDataPermission policy statements resulting

from the translation of Servlet `user-data-constraint` elements are an example of such permissions. A container must use one of the following techniques to check an instance of a permission for which policy is defined independent of AccessControlContext.

- The container calls `AccessControlContext.checkPermission` with the permission being checked as argument. The call to `checkPermission` may be made on any AccessControlContext. If `checkPermission` throws an AccessControlException, the permission is not granted. Otherwise the permission is granted.

- The container calls `AccessController.checkPermission` with the permission being checked. The value of the current thread's AccessControlContext is irrelevant in the access determination. If `checkPermission` throws an AccessControlException, the checked permission is not granted. Otherwise the permission is not granted.

- The container calls `SecurityManager.checkPermission` with the permission being checked. If `checkPermission` throws an AccessControlException, the checked permission is not granted. Otherwise the permission is granted.

- The J2EE 1.4 container calls `Policy.implies` with two arguments; the permission being checked and a ProtectionDomain that need not be constructed with principals. The checked permission is granted if Policy.implies returns true. Otherwise, the permission is not granted.

- The J2EE 1.4 container calls `java.security.Policy.getPermissions` with a ProtectionDomain that need not be constructed with principals. The container must call the `implies` method on the returned PermissionCollection using the permission being checked as argument. The checked permission is granted if the PermissionCollection implies it. Otherwise, the permission is not granted. This technique is supported but not recommended.

- The J2EE 1.3 container calls `javax.security.auth.Policy.getPermissions` to determine the collection of permissions granted independent of AccessControlContext. The Subject in the call to `getPermissions` may be null. The container must call

the `implies` method on the returned PermissionCollection using the permission being checked as argument. The checked permission is granted if the PermissionCollection implies it. Otherwise, the permission is not granted. This technique is supported but not recommended.

Prior to using any of the techniques described in this section, the container must have established a policy context identifier as defined in Section 4.6, "Setting the Policy Context".

4.8 Checking the Caller for a Permission

A container must determine if the caller has been granted a permission by evaluating the permission in the context of an AccessControlContext, ProtectionDomain, or Subject containing the principals of (only) the caller[2]. If the caller's identity has been asserted or vouched for by a trusted authority (other than the caller), the principals of the authority must not be included in the principals of the caller. A container must use one of the following techniques to determine if a permission has been granted to the caller.

- The container calls `AccessControlContext.checkPermission` with the permission as argument. The call to `checkPermission` must be made on an AccessControlContext that contains the principals of the caller. If checkPermission throws an AccessControlException, the permission is not granted to the caller. Otherwise the permission is granted.

- The container calls `AccessController.checkPermission` with the permission as argument. The AccessControlContext associated with the thread on which the call to `checkPermission` is made must contain the principals of the caller. If `checkPermission` throws an AccessControlException, the permission is not granted to the caller. Otherwise the permission is granted.

- The container calls `SecurityManager.checkPermission` with the permission as argument. The AccessControlContext associated with the thread on which the call to `checkPermission` is made must contain the principals

[2] Section 4.12, "Optimization of Permission Evaluations" allows containers to reuse granted results obtained for unauthenticated callers (i.e. with no principals) to authorize, independent of caller identity, permissions implied by such results.

of the caller. If checkPermission throws an AccessControlException, the permission is not granted to the caller. Otherwise the permission is granted.

- The J2EE 1.4 container calls `Policy.implies` with two arguments; the permission being checked and a ProtectionDomain constructed with the principals of the caller. The boolean result returned by `Policy.implies` indicates whether or not the permission has been granted to the caller.

- The J2EE 1.4 container calls `java.security.Policy.getPermissions` with an argument ProtectionDomain that was constructed with the principals of the caller. The container must call the `implies` method on the returned PermissionCollection using the permission being checked as argument. If the PermissionCollection implies the permission being tested, the permission has been granted to the caller. Otherwise it has not. This technique is supported but not recommended.[3]

- The J2EE 1.3 container calls `javax.security.auth.Policy.getPermissions` with an argument Subject containing the principals of the caller.The container must call the `implies` method on the returned PermissionCollection using the permission being checked as argument. If the PermissionCollection implies the permission being tested, the permission has been granted to the caller. Otherwise it has not. This technique is supported but not recommended.

Prior to using any of the techniques described in this section, the container must have established a policy context identifier as defined in Section 4.6, “Setting the Policy Context”.

4.9 Missing Policy Contexts

A Policy provider must return that a tested permission has not been granted if it acquires a non-null policy context identifier by calling getContextID on the PolicyContext class and the `inService` method of the `PolicyConfigurationFactory` associated with the provider would return false if called with the policy context identifier.

---

[3] Not all policy systems support this query. Also, the Policy provider does not see the permission being checked, and therefore cannot use the permission to identify when to invoke a particular policy context handler.

4.10 Default Policy Context

The default policy context contains the policy statements that apply to the JRE independent of the policy contexts defined as the result of the deployment of modules or applications in containers. The policy context identifier of the default policy context is the *null* value. The default policy context is never linked to another PolicyConfiguration, and as such does not share the principal-to-role mapping of any other policy context.

A Policy provider must include the policy statements of the default policy context in every access determination it performs. A Policy provider that either does not call PolicyContext.getContexdID, or does so and acquires the identifier of the default policy context, must use only the policy statements of the default policy context to perform its access determination

4.11 Policy Compatibility Requirements

To be compatible with this contract, every JRE of a J2EE 1.4 application server must perform all of the policy decisions defined by this contract by interacting with the `java.security.Policy` instance available in the JRE via the `java.security.Policy.getPolicy` method. Every JRE of a J2EE 1.3 application server must perform all of the policy decisions defined by this contract by interacting with the `javax.security.auth.Policy` instance available in the JRE via the `javax.security.auth.getPolicy` method.

If an application server or JRE employs a custom SecurityManager, the necessary reliance on Policy object may be accomplished by ensuring that the custom SecurityManager relies on the appropriate (as defined above) Policy object for all of the policy decisions defined by this contract.

4.12 Optimization of Permission Evaluations

Containers may employ the following optimizations (based on reuse) when the result obtained by repeating the evaluation will not differ from the previous result or when the time since the previous evaluation is less than the container's threshold for being effected by policy changes:

- Containers may reuse an authorization result obtained from a previous equivalent permission evaluation.
- Containers may reuse an authorization result obtained for an unauthenticated caller (i.e. a caller with no principals) performed as defined in Section 4.8, "Checking the Caller for a Permission" to grant, independent of caller identity, any permission implied by the unauthenticated result.

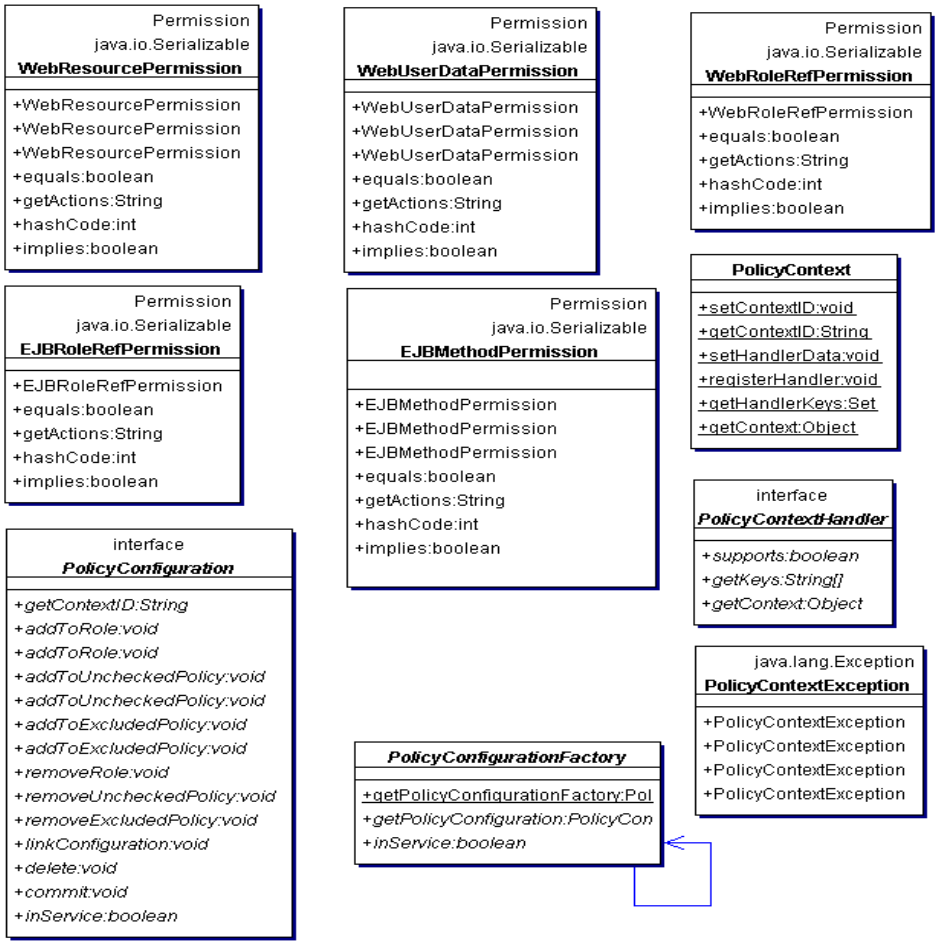
This specification does not prescribe how a container determines when a repeated evaluation will return the same result. That said, one way that containers could make this determination is if they are, and can determine if they will be, notified of policy changes and if they can establish that their policy provider does not employ additional context (such as could be acquired by calling a `PolicyContextHandler`) in its policy evaluations.

Common practice for containers to receive such notification could be for them to register to the `"java.security.Policy.supportsReuse"` key a `PolicyContextHandler` and for the container to determine if its provider will notify it of policy changes by making a test call to the provider's `refresh` method. Only a provider that is compatible with the optimizations described above (including because it does not employ additional context in its policy evaluations) may deliver notice of policy changes by activating this handler when its `refresh` method is called.

CHAPTER 5

API

Figure JSR115.5.1 javax.security.jacc Class Diagrams



Package

javax.security.jacc

Description

This package contains the Java Authorization Contract for Containers API

Class Summary

Interfaces

PolicyConfiguration	The methods of this interface are used by containers to create policy statements in a Policy provider.
PolicyContextHandler	This interface defines the methods that must be implemented by handlers that are to be registered and activated by the `PolicyContext` class.

Classes

EJBMethodPermission	Class for EJB method permissions.
EJBRoleRefPermission	Class for EJB *isCallerInRole (String reference)* permissions.
PolicyConfiguration-Factory	Abstract factory and finder class for obtaining the instance of the class that implements the PolicyConfigurationFactory of a provider.
PolicyContext	This utility class is used by containers to communicate policy context identifiers and other policy relevant context to `Policy` providers.
WebResourcePermission	Class for Servlet web resource permissions.
WebRoleRefPermission	Class for Servlet *isUserInRole (String reference)* permissions.
WebUserDataPermission	Class for Servlet Web user data permissions.

Exceptions

PolicyContextExcep-tion	This checked exception is thrown by implementations of the `javax.security.jacc.PolicyConfiguration` Interface, the `javax.security.jacc.PolicyConfigurationFactory` abstract class, the `javax.security.jacc.PolicyContext` utility class, and implementations of the `javax.security.jacc.PolicyContextException` Interface.

javax.security.jacc

EJBMethodPermission

Declaration

```
public final class EJBMethodPermission extends java.security.Permission implements
    java.io.Serializable

java.lang.Object
  |
  +--java.security.Permission
        |
        +--javax.security.jacc.EJBMethodPermission
```

All Implemented Interfaces: `java.security.Guard`, `java.io.Serializable`

Description

Class for EJB method permissions.

The name of an EJBMethodPermission contains the value of the ejb-name element in the application's deployment descriptor that identifies the target EJB.

The actions of an EJBMethodPermission identifies the methods of the EJB to which the permission applies.

Implementations of this class MAY implement newPermissionCollection or inherit its implementation from the super class.

See Also: `java.security.Permission`

Member Summary

Constructors

`EJBMethodPermission(java.lang.String name, java.lang.String actions)`
Creates a new EJBMethodPermission with the specified name and actions.

`EJBMethodPermission(java.lang.String EJBName, java.lang.String methodInterface, java.lang.reflect.Method method)`
Creates a new EJBMethodPermission with name corresponding to the EJBName and actions composed from methodInterface, and the Method object.

`EJBMethodPermission(java.lang.String EJBName, java.lang.String methodName, java.lang.String methodInterface, java.lang.String methodParams)`
Creates a new EJBMethodPermission with name corresponding to the EJBName and actions composed from methodName, methodInterface, and methodParams.

Methods

`boolean`	`equals(java.lang.Object o)` Checks two EJBMethodPermission objects for equality.
`java.lang.String`	`getActions()` Returns a String containing a canonical representation of the actions of this EJBMethodPermission.

Member Summary

int	hashCode() Returns the hash code value for this EJBMethodPermission.
boolean	implies(java.security.Permission permission) Determines if the argument Permission is “implied by” this EJBMethodPermission.

Inherited Member Summary

Methods inherited from class `Object`

`clone(), finalize(), getClass(), notify(), notifyAll(), wait(long, int), wait(long, int), wait(long, int)`

Methods inherited from class `Permission`

`checkGuard(Object), getName(), newPermissionCollection(), toString()`

Constructors

EJBMethodPermission(String, String)

```
public EJBMethodPermission(java.lang.String name, java.lang.String actions)
```

Creates a new EJBMethodPermission with the specified name and actions.

The name contains the value of the ejb-name element corresponding to an EJB in the application’s deployment descriptor.

The actions contains a methodSpec. The syntax of the actions parameter is defined as follows:

```
methodNameSpec ::= methodName | emptyString
methodInterfaceName ::= String
methodInterfaceSpec ::= methodInterfaceName | emptyString
typeName ::= typeName | typeName []
methodParams ::= typeName | methodParams comma typeName
methodParamsSpec ::= emptyString | methodParams
methodSpec ::= null |
     methodNameSpec |
     methodNameSpec comma methodInterfaceName |
     methodNameSpec comma methodInterfaceSpec comma methodParamsSpec
```

A MethodInterfaceName is a non-empty String and should contain a method-intf value as defined for use in EJB deployment descriptors. An implementation must be flexible such that it supports additional interface names especially if they are standardized by the EJB Specification. The EJB Specification currently defines the following method-intf values:

```
{ “Home”, “LocalHome”, “Remote”, “Local”, “ServiceEndpoint” }
```

A null or empty string methodSpec indicates that the permission applies to all methods of the EJB. A methodSpec with a methodNameSpec of the empty string matches all methods of the EJB that match the methodInterface and methodParams elements of the methodSpec.

A methodSpec with a methodInterfaceSpec of the empty string matches all methods of the EJB that match the methodNameSpec and methodParamsSpec elements of the methodSpec.

A methodSpec without a methodParamsSpec matches all methods of the EJB that match the methodNameSpec and methodInterface elements of the methodSpec.

The order of the typeNames in methodParams array must match the order of occurence of the corresponding parameters in the method signature of the target method(s). Each typeName in the methodParams must contain the canonical form of the corresponding parameter's typeName as defined by the getActions method. A methodSpec with an empty methodParamsSpec matches all 0 argument methods of the EJB that match the methodNameSpec and methodInterfaceSpec elements of the methodSpec.

Parameters:

`name` - of the EJB to which the permission pertains.

`actions` - identifies the methods of the EJB to which the permission pertains.

EJBMethodPermission(String, String, Method)

```
public EJBMethodPermission(java.lang.String EJBName, java.lang.String methodInterface,
            java.lang.reflect.Method method)
```

Creates a new EJBMethodPermission with name corresponding to the EJBName and actions composed from methodInterface, and the Method object.

A container uses this constructor prior to checking if a caller has permission to call the method of an EJB.

Parameters:

`EJBName` - The string representation of the name of the EJB as it appears in the corresponding ejb-name element in the deployment descriptor.

`methodInterface` - A string that may be used to specify the EJB interface to which the permission pertains. A value of null or "", indicates that the permission pertains to all methods that match the other parameters of the permission specification without consideration of the interface they occur on.

`method` - an instance of the Java.lang.reflect.Method class corresponding to the method that the container is trying to determine whether the caller has permission to access. This value must not be null.

EJBMethodPermission(String, String, String, String[])

```
public EJBMethodPermission(java.lang.String EJBName, java.lang.String methodName,
            java.lang.String methodInterface, java.lang.String[] methodParams)
```

Creates a new EJBMethodPermission with name corresponding to the EJBName and actions composed from methodName, methodInterface, and methodParams.

Parameters:

`EJBName` - The string representation of the name of the EJB as it appears in the corresponding ejb-name element in the deployment descriptor.

`methodName` - A string that may be used to indicate the method of the EJB to which the permission pertains. A value of null or "" indicates that the permission pertains to all methods that match the other parameters of the permission specification without consideration of method name.

`methodInterface` - A string that may be used to specify the EJB interface to which the permission pertains. A value of null or "", indicates that the permission pertains to all methods that match the other parameters of the permission specification without consideration of the interface they occur on.

`methodParams` - An array of strings that may be used to specify (by typeNames) the parameter signature of the target methods. The order of the typeNames in methodParams array must match the order of occurence of the corresponding parameters in the method signature of the target method(s).

Each typeName in the methodParams array must contain the canonical form of the corresponding parameter's typeName as defined by the getActions method. An empty methodParams array is used to represent a method signature with no arguments. A value of null indicates that the permission pertains to all methods that match the other parameters of the permission specification without consideration of method signature.

Methods

equals(Object)

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

Checks two EJBMethodPermission objects for equality. EJBMethodPermission objects are equivalent if they have case sensitive equivalent name and actions values.

Two Permission objects, P1 and P2, are equivalent if and only if P1.implies(P2) && P2.implies(P1).

Overrides: `equals` in class `Permission`

Parameters:
`o` - the EJBMethodPermission object being tested for equality with this EJBMethodPermission

Returns: true if the argument EJBMethodPermission object is equivalent to this EJBMethodPermission.

getActions()

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

Returns a String containing a canonical representation of the actions of this EJBMethodPermission. The Canonical form of the actions of an EJBMethodPermission is described by the following syntax description.

```
methodNameSpec ::= methodName | emptyString
methodInterfaceName ::= String
methodInterfaceSpec ::= methodInterfaceName | emptyString
typeName ::= typeName | typeName []
methodParams ::= typeName | methodParams comma typeName
methodParamsSpec ::= emptyString | methodParams
methodSpec ::= null |
     methodName |
     methodNameSpec comma methodInterfaceName |
     methodNameSpec comma methodInterfaceSpec comma methodParamsSpec
```

The canonical form of each typeName must begin with the fully qualified Java name of the corresponding parameter's type. The canonical form of a typeName for an array parameter is the fully qualified Java name of the array's component type followed by as many instances of the string "[]" as there are dimensions to the array. No additional characters (e.g. blanks) may occur in the canonical form.

A MethodInterfaceName is a non-empty String and should contain a method-intf value as defined for use in EJB deployment descriptors. An implementation must be flexible such that it supports additional interface names especially if they are standardized by the EJB Specification. The EJB Specification currently defines the following method-intf values:

```
{ "Home", "LocalHome", "Remote", "Local", "ServiceEndpoint" }
```

Overrides: `getActions` in class `Permission`

Returns: a String containing the canonicalized actions of this EJBMethodPermission.

hashCode()

```
public int hashCode()
```

Returns the hash code value for this EJBMethodPermission. The properties of the returned hash code must be as follows:

- During the lifetime of a Java application, the hashCode method must return the same integer value every time it is called on a EJBMethodPermission object. The value returned by hashCode for a particular EJBMethodPermission need not remain consistent from one execution of an application to another.
- If two EJBMethodPermission objects are equal according to the equals method, then calling the hashCode method on each of the two Permission objects must produce the same integer result (within an application).

Overrides: `hashCode` in class `Permission`

Returns: the integer hash code value for this object.

implies(Permission)

```
public boolean implies(java.security.Permission permission)
```

Determines if the argument Permission is "implied by" this EJBMethodPermission. For this to be the case,

- The argument must be an instanceof EJBMethodPermission
- with name equivalent to that of this EJBMethodPermission, and
- the methods to which the argument permission applies (as defined in its actions) must be a subset of the methods to which this EJBMethodPermission applies (as defined in its actions).

The argument permission applies to a subset of the methods to which this permission applies if all of the following conditions are met.

- the method name component of the methodNameSpec of this permission is null, the empty string, or equivalent to the method name of the argument permission, and
- the method interface component of the methodNameSpec of this permission is null, the empty string, or equivalent to the method interface of the argument permission, and
- the method parameter list component of the methodNameSpec of this permission is null, the empty string, or equivalent to the method parameter list of the argument permission.

The name and actions comparisons described above are case sensitive.

Overrides: `implies` in class `Permission`

Parameters:
`permission` - "this" EJBMethodPermission is checked to see if it implies the argument permission.

Returns: true if the specified permission is implied by this object, false if not.

javax.security.jacc

EJBRoleRefPermission

Declaration

```
public final class EJBRoleRefPermission extends java.security.Permission implements
    java.io.Serializable

java.lang.Object
  |
  +--java.security.Permission
        |
        +--javax.security.jacc.EJBRoleRefPermission
```

All Implemented Interfaces: `java.security.Guard`, `java.io.Serializable`

Description

Class for EJB *isCallerInRole (String reference)* permissions. An EJBRoleRefPermission is a named permission and has actions.

The name of an EJBRoleRefPermission contains the value of the ejb-name element in the application's deployment descriptor that identifies the EJB in whose context the permission is being evalutated.

the actions of an EJBRoleRefPermission identifies the role reference to which the permission applies. An EJBRoleRefPermission is checked to determine if the subject is a member of the role identified by the reference.

Implementations of this class MAY implement newPermissionCollection or inherit its implementation from the super class.

See Also: `java.security.Permission`

Member Summary

Constructors

	`EJBRoleRefPermission(java.lang.String name, java.lang.String actions)` Creates a new EJBRoleRefPermission with the specified name and actions.

Methods

`boolean`	`equals(java.lang.Object o)` Checks two EJBRoleRefPermission objects for equality.
`java.lang.String`	`getActions()` Returns a canonical String representation of the actions of this EJBRoleRef-Permission.
`int`	`hashCode()` Returns the hash code value for this EJBRoleRefPermission.
`boolean`	`implies(java.security.Permission permission)` Determines if the argument Permission is "implied by" this EJBRoleRefPermission.

Inherited Member Summary

Methods inherited from class `Object`

`clone(), finalize(), getClass(), notify(), notifyAll(), wait(long, int), wait(long, int), wait(long, int)`

Methods inherited from class `Permission`

`checkGuard(Object), getName(), newPermissionCollection(), toString()`

Constructors

EJBRoleRefPermission(String, String)

`public` **`EJBRoleRefPermission`**`(java.lang.String name, java.lang.String actions)`

Creates a new EJBRoleRefPermission with the specified name and actions.

Parameters:

`name` - the ejb-name that identifies the EJB in whose context the role references are to be evaluated.

`actions` - identifies the role reference to which the permission pertains. The role reference is scoped to the EJB identified in the name parameter. The value of the role reference must not be `null` or the empty string.

Methods

equals(Object)

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

Checks two EJBRoleRefPermission objects for equality. EJBRoleRefPermission objects are equivalent if they have case equivalent name and actions values.

Two Permission objects, P1 and P2, are equivalent if and only if P1.implies(P2) && P2.implies(P1).

Overrides: `equals` in class `Permission`

Parameters:

`o` - the EJBRoleRefPermission object being tested for equality with this EJBRoleRefPermission.

Returns: true if the argument EJBRoleRefPermission object is equivalent to this EJBRoleRefPermission.

getActions()

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

Returns a canonical String representation of the actions of this EJBRoleRefPermission.

Overrides: `getActions` in class `Permission`

Returns: a String containing the canonicalized actions of this EJBRoleRefPermission.

hashCode()

`public int` **`hashCode`**`()`

Returns the hash code value for this EJBRoleRefPermission. The properties of the returned hash code must be as follows:

- During the lifetime of a Java application, the hashCode method must return the same integer value, every time it is called on a EJBRoleRefPermission object. The value returned by hashCode for a particular EJBRoleRefPermission need not remain consistent from one execution of an application to another.
- If two EJBRoleRefPermission objects are equal according to the equals method, then calling the hashCode method on each of the two Permission objects must produce the same integer result (within an application).

Overrides: `hashCode` in class `Permission`

Returns: the integer hash code value for this object.

implies(Permission)

```
public boolean implies(java.security.Permission permission)
```

Determines if the argument Permission is “implied by” this EJBRoleRefPermission. For this to be the case,

The argument must be an instanceof EJBRoleRefPermission with name equivalent to that of this EJBRoleRefPermission, and with the role reference equivalent to that of this EJBRoleRefPermission applies.

The name and actions comparisons described above are case sensitive.

Overrides: `implies` in class `Permission`

Parameters:

`permission` - “this” EJBRoleRefPermission is checked to see if it implies the argument permission.

Returns: true if the specified permission is implied by this object, false if not.

javax.security.jacc

PolicyConfiguration

Declaration

```
public interface PolicyConfiguration
```

Description

The methods of this interface are used by containers to create policy statements in a Policy provider. An object that implements the PolicyConfiguration interface provides the policy statement configuration interface for a corresponding policy context within the corresponding Policy provider.

The life cycle of a policy context is defined by three states; “open”, “inService”, and “deleted”. A policy context is in one of these three states.

A policy context in the “open” state is in the process of being configured, and may be operated on by any of the methods of the PolicyConfiguration interface. A policy context in the “open” state must not be assimilated at `Policy.refresh` into the policy statements used by the Policy provider in performing its access decisions. In order for the policy statements of a policy context to be assimilated by the associated provider, the policy context must be in the “inService” state. A policy context in the “open” state is transitioned to the “inService” state by calling the commit method.

A policy context in the “inService” state is available for assimilation into the policy statements being used to perform access decisions by the associated Policy provider. Providers assimilate policy contexts containing policy statements when the refresh method of the provider is called. When a provider’s refresh method is called, it must assimilate only those policy contexts whose state is “inService” and it must ensure that the policy statements put into service for each policy context are only those defined in the context at the time of the call to refresh. A policy context in the “inService” state is not available for additional configuration and may be returned to the “open” state by calling the getPolicyConfiguration method of the PolicyConfigurationFactory.

A policy context in the “deleted” state is neither available for configuration, nor is it available for assimilation into the Provider. A policy context whose state is “deleted” may be reclaimed for subsequent processing by calling the getPolicyConfiguration method of the associated PolicyConfigurationFactory. A “deleted” policy context is transitioned to the “open” state when it it returned as a result of a call to getPolicyConfiguration.

The following table captures the correspondence between the policy context life cycle and the methods of the PolicyConfiguration interface. The rightmost 3 columns of the table correspond to the PolicyConfiguration state identified at the head of the column. The values in the cells of these columns indicate the next state resulting from a call to the method identifed in the leftmost column of the corresponding row, or that calling the method is unsupported in the state represented by the column (in which case the state will remain unchanged).

PolicyConfiguration State Table Method	Current State to Next State		
	deleted	open	inService
addToExcludedPolicy	Unsupported Operation	open	Unsupported Operation
addToRole	Unsupported Operation	open	Unsupported Operation

addToUncheckedPolicy	Unsupported Operation	open	Unsupported Operation
commit	Unsupported Operation	inService	inService
delete	deleted	deleted	deleted
getContextID	deleted	open	inService
inService	deleted	open	inService
linkConfiguration	Unsupported Operation	open	Unsupported Operation
removeExcludedPolicy	Unsupported Operation	open	Unsupported Operation
removeRole	Unsupported Operation	open	Unsupported Operation
removeUncheckedPolicy	Unsupported Operation	open	Unsupported Operation

For a provider implementation to be compatible with multi-threaded environments, it may be necessary to synchronize the refresh method of the provider with the methods of its PolicyConfiguration interface and with the getPolicyConfiguration and inService methods of its PolicyConfigurationFactory.

See Also: `java.security.Permission`, `java.security.PermissionCollection`, `PolicyContextException`, `PolicyConfigurationFactory`

Member Summary

Methods

`void` `addToExcludedPolicy(java.security.Permission permission)`
Used to add a single excluded policy statement to this PolicyConfiguration.

`void` `addToExcludedPolicy(java.security.PermissionCollection permissions)`
Used to add excluded policy statements to this PolicyConfiguration.

`void` `addToRole(java.lang.String roleName, java.security.Permission permission)`
Used to add a single permission to a named role in this PolicyConfiguration.

`void` `addToRole(java.lang.String roleName, java.security.PermissionCollection permissions)`
Used to add permissions to a named role in this PolicyConfiguration.

`void` `addToUncheckedPolicy(java.security.Permission permission)`
Used to add a single unchecked policy statement to this PolicyConfiguration.

`void` `addToUncheckedPolicy(java.security.PermissionCollection permissions)`
Used to add unchecked policy statements to this PolicyConfiguration.

`void` `commit()`
This method is used to set to “inService” the state of the policy context whose interface is this PolicyConfiguration Object.

`void` `delete()`
Causes all policy statements to be deleted from this PolicyConfiguration and sets its internal state such that calling any method, other than delete, getContextID, or inService on the PolicyConfiguration will be rejected and cause an UnsupportedOperationException to be thrown.

`java.lang.String` `getContextID()`
This method returns this object’s policy context identifier.

Member Summary	
boolean	inService() This method is used to determine if the policy context whose interface is this PolicyConfiguration Object is in the "inService" state.
void	linkConfiguration(PolicyConfiguration link) Creates a relationship between this configuration and another such that they share the same principal-to-role mappings.
void	removeExcludedPolicy() Used to remove any excluded policy statements from this PolicyConfiguration.
void	removeRole(java.lang.String roleName) Used to remove a role and all its permissions from this PolicyConfiguration.
void	removeUncheckedPolicy() Used to remove any unchecked policy statements from this PolicyConfiguration.

Methods

addToExcludedPolicy(Permission)

```
public void addToExcludedPolicy(java.security.Permission permission)
            throws PolicyContextException
```

Used to add a single excluded policy statement to this PolicyConfiguration.

Parameters:

`permission` - the permission to be added to the excluded policy statements.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.UnsupportedOperationException` - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the addToExcludedPolicy method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

addToExcludedPolicy(PermissionCollection)

```
public void addToExcludedPolicy(java.security.PermissionCollection permissions)
            throws PolicyContextException
```

Used to add excluded policy statements to this PolicyConfiguration.

Parameters:

`permissions` - the collection of permissions to be added to the excluded policy statements. The collection may be either a homogenous or heterogenous collection.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.UnsupportedOperationException` - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

PolicyContextException - if the implementation throws a checked exception that has not been accounted for by the addToExcludedPolicy method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

addToRole(String, Permission)

```
public void addToRole(java.lang.String roleName, java.security.Permission permission)
            throws PolicyContextException
```

Used to add a single permission to a named role in this PolicyConfiguration. If the named Role does not exist in the PolicyConfiguration, it is created as a result of the call to this function.

It is the job of the Policy provider to ensure that all the permissions added to a role are granted to principals "mapped to the role".

Parameters:

roleName - the name of the Role to which the permission is to be added.

permission - the permission to be added to the role.

Throws:

java.lang.SecurityException - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

java.lang.UnsupportedOperationException - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

PolicyContextException - if the implementation throws a checked exception that has not been accounted for by the addToRole method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

addToRole(String, PermissionCollection)

```
public void addToRole(java.lang.String roleName,
            java.security.PermissionCollection permissions)
            throws PolicyContextException
```

Used to add permissions to a named role in this PolicyConfiguration. If the named Role does not exist in the PolicyConfiguration, it is created as a result of the call to this function.

It is the job of the Policy provider to ensure that all the permissions added to a role are granted to principals "mapped to the role".

Parameters:

roleName - the name of the Role to which the permissions are to be added.

permissions - the collection of permissions to be added to the role. The collection may be either a homogenous or heterogenous collection.

Throws:

java.lang.SecurityException - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

java.lang.UnsupportedOperationException - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

PolicyContextException - if the implementation throws a checked exception that has not been accounted for by the addToRole method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

addToUncheckedPolicy(Permission)

```
public void addToUncheckedPolicy(java.security.Permission permission)
            throws PolicyContextException
```

Used to add a single unchecked policy statement to this PolicyConfiguration.

Parameters:

`permission` - the permission to be added to the unchecked policy statements.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.UnsupportedOperationException` - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the addToUncheckedPolicy method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

addToUncheckedPolicy(PermissionCollection)

```
public void addToUncheckedPolicy(java.security.PermissionCollection permissions)
            throws PolicyContextException
```

Used to add unchecked policy statements to this PolicyConfiguration.

Parameters:

`permissions` - the collection of permissions to be added as unchecked policy statements. The collection may be either a homogenous or heterogenous collection.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.UnsupportedOperationException` - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the addToUncheckedPolicy method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

commit()

```
public void commit()
            throws PolicyContextException
```

This method is used to set to "inService" the state of the policy context whose interface is this PolicyConfiguration Object. Only those policy contexts whose state is "inService" will be included in the policy contexts processed by the Policy.refresh method. A policy context whose state is "inService" may be returned to the "open" state by calling the getPolicyConfiguration method of the PolicyConfiguration factory with the policy context identifier of the policy context.

When the state of a policy context is "inService", calling any method other than commit, delete, getContextID, or inService on its PolicyConfiguration Object will cause an UnsupportedOperationException to be thrown.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.UnsupportedOperationException` - if the state of the policy context whoseinterface is this PolicyConfiguration Object is "deleted" when this method is called. This exception is also thrown when completing the commit would cause there to be two or more inService and linked policy contexts with different principal-to-role mappings.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the commit method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

delete()

```
public void delete()
            throws PolicyContextException
```

Causes all policy statements to be deleted from this PolicyConfiguration and sets its internal state such that calling any method, other than delete, getContextID, or inService on the PolicyConfiguration will be rejected and cause an UnsupportedOperationException to be thrown.

This operation has no affect on any linked PolicyConfigurations other than removing any links involving the deleted PolicyConfiguration.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the delete method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

getContextID()

```
public java.lang.String getContextID()
            throws PolicyContextException
```

This method returns this object's policy context identifier.

Returns: this object's policy context identifier.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the getContextID method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

inService()

```
public boolean inService()
            throws PolicyContextException
```

This method is used to determine if the policy context whose interface is this PolicyConfiguration Object is in the "inService" state.

Returns: true if the state of the associated policy context is "inService"; false otherwise.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the inService method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

linkConfiguration(PolicyConfiguration)

```
public void linkConfiguration(javax.security.jacc.PolicyConfiguration link)
            throws PolicyContextException
```

Creates a relationship between this configuration and another such that they share the same principal-to-role mappings. PolicyConfigurations are linked to apply a common principal-to-role mapping to multiple seperately manageable PolicyConfigurations, as is required when an application is composed of multiple modules.

Note that the policy statements which comprise a role, or comprise the excluded or unchecked policy collections in a PolicyConfiguration are unaffected by the configuration being linked to another.

Parameters:

`link` - a reference to a different PolicyConfiguration than this PolicyConfiguration.

The relationship formed by this method is symetric, transitive and idempotent. If the argument PolicyConfiguration does not have a different Policy context identifier than this PolicyConfiguration no relationship is formed, and an exception, as described below, is thrown.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.UnsupportedOperationException` - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

`java.lang.IllegalArgumentException` - if called with an argument PolicyConfiguration whose Policy context is equivalent to that of this PolicyConfiguration.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the linkConfiguration method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

removeExcludedPolicy()

```
public void removeExcludedPolicy()
            throws PolicyContextException
```

Used to remove any excluded policy statements from this PolicyConfiguration.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.UnsupportedOperationException` - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the removeExcludedPolicy method signature. The exception thrown by the

implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

removeRole(String)

```
public void removeRole(java.lang.String roleName)
            throws PolicyContextException
```

Used to remove a role and all its permissions from this PolicyConfiguration.

Parameters:

`roleName` - the name of the Role to remove from this PolicyConfiguration.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.UnsupportedOperationException` - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the removeRole method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

removeUncheckedPolicy()

```
public void removeUncheckedPolicy()
            throws PolicyContextException
```

Used to remove any unchecked policy statements from this PolicyConfiguration.

Throws:

`java.lang.SecurityException` - if called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.UnsupportedOperationException` - if the state of the policy context whose interface is this PolicyConfiguration Object is "deleted" or "inService" when this method is called.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the removeUncheckedPolicy method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

javax.security.jacc

PolicyConfigurationFactory

Declaration

```
public abstract class PolicyConfigurationFactory

java.lang.Object
  |
  +--javax.security.jacc.PolicyConfigurationFactory
```

Description

Abstract factory and finder class for obtaining the instance of the class that implements the PolicyConfigurationFactory of a provider. The factory will be used to instantiate PolicyConfiguration objects that will be used by the deployment tools of the container to create and manage policy contexts within the Policy Provider.

Implementation classes must have a public no argument constructor that may be used to create an operational instance of the factory implementation class.

See Also: `java.security.Permission`, PolicyConfiguration, PolicyContextException

Member Summary

Constructors

	`PolicyConfigurationFactory()`

Methods

abstract PolicyConfiguration	`getPolicyConfiguration(java.lang.String contextID, boolean remove)` This method is used to obtain an instance of the provider specific class that implements the PolicyConfiguration interface that corresponds to the identified policy context within the provider.
static PolicyConfigurationFactory	`getPolicyConfigurationFactory()` This static method uses a system property to find and instantiate (via a public constructor) a provider specific factory implementation class.
abstract boolean	`inService(java.lang.String contextID)` This method determines if the identified policy context exists with state “inService” in the Policy provider associated with the factory.

Inherited Member Summary

Methods inherited from class `Object`

`clone()`, `equals(Object)`, `finalize()`, `getClass()`, `hashCode()`, `notify()`, `notifyAll()`, `toString()`, `wait(long, int)`, `wait(long, int)`, `wait(long, int)`

Constructors

PolicyConfigurationFactory()

```
public PolicyConfigurationFactory()
```

Methods

getPolicyConfiguration(String, boolean)

```
public abstract javax.security.jacc.PolicyConfiguration
            getPolicyConfiguration(java.lang.String contextID, boolean remove)
            throws PolicyContextException
```

This method is used to obtain an instance of the provider specific class that implements the PolicyConfiguration interface that corresponds to the identified policy context within the provider. The methods of the PolicyConfiguration interface are used to define the policy statements of the identified policy context.

If at the time of the call, the identified policy context does not exist in the provider, then the policy context will be created in the provider and the Object that implements the context's PolicyConfiguration Interface will be returned. If the state of the identified context is "deleted" or "inService" it will be transitioned to the "open" state as a result of the call. The states in the lifecycle of a policy context are defined by the PolicyConfiguration interface.

For a given value of policy context identifier, this method must always return the same instance of PolicyConfiguration and there must be at most one actual instance of a PolicyConfiguration with a given policy context identifier (during a process context).

To preserve the invariant that there be at most one PolicyConfiguration object for a given policy context, it may be necessary for this method to be thread safe.

Parameters:

`contextID` - A String identifying the policy context whose PolicyConfiguration interface is to be returned. The value passed to this parameter must not be null.

`remove` - A boolean value that establishes whether or not the policy statements of an existing policy context are to be removed before its PolicyConfiguration object is returned. If the value passed to this parameter is true, the policy statements of an existing policy context will be removed. If the value is false, they will not be removed.

Returns: an Object that implements the PolicyConfiguration Interface matched to the Policy provider and corresponding to the identified policy context.

Throws:

`java.lang.SecurityException` - when called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the getPolicyConfiguration method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

getPolicyConfigurationFactory()

```
public static javax.security.jacc.PolicyConfigurationFactory
            getPolicyConfigurationFactory()
            throws ClassNotFoundException, PolicyContextException
```

This static method uses a system property to find and instantiate (via a public constructor) a provider specific factory implementation class. The name of the provider specific factory implementation class is obtained from the value of the system property,

```
javax.security.jacc.PolicyConfigurationFactory.provider.
```

Returns: the singleton instance of the provider specific PolicyConfigurationFactory implementation class.

Throws:

`java.lang.SecurityException` - when called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`java.lang.ClassNotFoundException` - when the class named by the system property could not be found including because the value of the system property has not be set.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the getPolicyConfigurationFactory method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException

inService(String)

```
public abstract boolean inService(java.lang.String contextID)
            throws PolicyContextException
```

This method determines if the identified policy context exists with state "inService" in the Policy provider associated with the factory.

Parameters:

`contextID` - A string identifying a policy context

Returns: true if the identified policy context exists within the provider and its state is "inService", false otherwise.

Throws:

`java.lang.SecurityException` - when called by an AccessControlContext that has not been granted the "setPolicy" SecurityPermission.

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the inService method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException.

javax.security.jacc

PolicyContext

Declaration

```
public final class PolicyContext

java.lang.Object
  |
  +--javax.security.jacc.PolicyContext
```

Description

This utility class is used by containers to communicate policy context identifiers and other policy relevant context to `Policy` providers. `Policy` providers use the policy context identifier to select the subset of policy to apply in access decisions.

The value of a policy context identifier is a `String` and each thread has an independently established policy context identifier. A container will establish the thread-scoped value of a policy context identifier by calling the static `setContextID` method. The value of a thread-scoped policy context identifier is available (to `Policy`) by calling the static `getContextID` method.

This class is also used by `Policy` providers to request additional thread-scoped policy relevant context objects from the calling container. Containers register container-specific `PolicyContext` handlers using the static `registerHandler` method. Handler registration is scoped to the class, such that the same handler registrations are active in all thread contexts. Containers may use the static method `setHandlerData` to establish a thread-scoped parameter that will be passed to handlers when they are activated by `Policy` providers. The static `getContext` method is used to activate a handler and obtain the corresponding context object.

The static accessor functions provided by this class allow per-thread policy context values to be established and communicated independent of a common reference to a particular PolicyContext instance.

The PolicyContext class may encapsulate static ThreadLocal instance variables to represent the policy context identifier and handler data values.

The Application server must bundle or install the PolicyContext class, and the containers of the application server must prevent the methods of the PolicyContext class from being called from calling contexts that are not authorized to call these methods. With the exception of the getContextID and GetHandlerKeys methods, containers must restrict and afford access to the methods of the PolicyContext class to calling contexts trusted by the container to perform container access decisions. The PolicyContext class may satisfy this requirement (on behalf of its container) by rejecting calls made from an AccessControlContext that has not been granted the “setPolicy” SecurityPermission, and by ensuring that Policy providers used to perform container access decisions are granted the “setPolicy” permission.

See Also: `PolicyContextHandler`

Member Summary
Methods

Member Summary

static java.lang.Object	getContext(java.lang.String key) This method may be used by a Policy provider to activate the PolicyContextHandler registered to the context object key and cause it to return the corresponding policy context object from the container.
static java.lang.String	getContextID() This static method returns the value of the policy context identifier associated with the thread on which the accessor is called.
static java.util.Set	getHandlerKeys() This method may be used to obtain the keys that identify the container specific context handlers registered by the container.
static void	registerHandler(java.lang.String key, PolicyContextHandler handler, boolean replace) Authorization protected method used to register a container specific PolicyContext handler.
static void	setContextID(java.lang.String contextID) Authorization protected method used to modify the value of the policy context identifier associated with the thread on which this method is called.
static void	setHandlerData(java.lang.Object data) Authorization protected method that may be used to associate a thread-scoped handler data object with the PolicyContext.

Inherited Member Summary

Methods inherited from class `Object`

`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), toString(), wait(long, int), wait(long, int), wait(long, int)`

Methods

getContext(String)

```
public static java.lang.Object getContext(java.lang.String key)
            throws PolicyContextException
```

This method may be used by a `Policy` provider to activate the `PolicyContextHandler` registered to the context object key and cause it to return the corresponding policy context object from the container. When this method activates a handler, it passes to the handler the context object key and the handler data associated with the calling thread.

Parameters:

`key` - a `String` that identifies the `PolicyContextHandler` to activate and the context object to be acquired from the handler. The value of this parameter must not be null.

Returns: the container and handler specific object containing the desired context. A `null` value is returned if the corresponding handler has been registered, and the value of the corresponding context is null.

Throws:

`java.lang.IllegalArgumentException` - if a `PolicyContextHandler` has not been registered for the key or the registered handler no longer supports the key.

`java.lang.SecurityException` - if the calling AccessControlContext is not authorized by the container to call this method.

`PolicyContextException` - if an operation by this method on the identified PolicyContextHandler causes it to throw a checked exception that is not accounted for in the signature of this method.

getContextID()

```
public static java.lang.String getContextID()
```

This static method returns the value of the policy context identifier associated with the thread on which the accessor is called.

Returns: The `String` (or `null`) policy context identifier established for the thread. This method must return the default policy context identifier, `null`, if the policy context identifier of the thread has not been set via `setContext` to another value.

Throws:

`java.lang.SecurityException` - if the calling AccessControlContext is not authorized by the container to call this method. Containers may choose to authorize calls to this method by any AccessControlContext.

getHandlerKeys()

```
public static java.util.Set getHandlerKeys()
```

This method may be used to obtain the keys that identify the container specific context handlers registered by the container.

Returns: A `Set`, the elements of which, are the `String` key values that identify the handlers that have been registered and therefore may be activated on the `PolicyContext`.

Throws:

`java.lang.SecurityException` - if the calling AccessControlContext is not authorized by the container to call this method. Containers may choose to authorize calls to this method by any AccessControlContext.

registerHandler(String, PolicyContextHandler, boolean)

```
public static void registerHandler(java.lang.String key,
            javax.security.jacc.PolicyContextHandler handler, boolean replace)
            throws PolicyContextException
```

Authorization protected method used to register a container specific `PolicyContext` handler. A handler may be registered to handle multiple keys, but at any time, at most one handler may be registered for a key.

Parameters:

`key` - a (case-sensitive) `String` that identifies the context object handled by the handler. The value of this parameter must not be null.

`handler` - an object that implements the `PolicyContextHandler` interface. The value of this parameter must not be null.

`replace` - this boolean value defines the behavior of this method if, when it is called, a `PolicyContextHandler` has already been registered to handle the same key. In that case, and if the value of this argument is `true`, the existing handler is replaced with the argument handler. If the value of this parameter is false the existing registration is preserved and an exception is thrown.

Throws:

`java.lang.IllegalArgumentException` - if the value of either of the handler or key arguments is null, or the value of the replace argument is `false` and a handler with the same key as the argument handler is already registered.

`java.lang.SecurityException` - if the calling AccessControlContext is not authorized by the container to call this method.

`PolicyContextException` - if an operation by this method on the argument PolicyContextHandler causes it to throw a checked exception that is not accounted for in the signature of this method.

setContextID(String)

```
public static void setContextID(java.lang.String contextID)
```

Authorization protected method used to modify the value of the policy context identifier associated with the thread on which this method is called.

Parameters:

`contextID` - a `String` that represents the value of the policy context identifier to be assigned to the PolicyContext for the calling thread. The value `null` is a legitimate value for this parameter.

Throws:

`java.lang.SecurityException` - if the calling AccessControlContext is not authorized by the container to call this method.

setHandlerData(Object)

```
public static void setHandlerData(java.lang.Object data)
```

Authorization protected method that may be used to associate a thread-scoped handler data object with the PolicyContext. The handler data object will be made available to handlers, where it can serve to supply or bind the handler to invocation scoped state within the container.

Parameters:

`data` - a container-specific object that will be associated with the calling thread and passed to any handler activated by a `Policy` provider (on the thread). The value `null` is a legitimate value for this parameter, and is the value that will be used in the activation of handlers if the `setHandlerData` has not been called on the thread.

Throws:

`java.lang.SecurityException` - if the calling AccessControlContext is not authorized by the container to call this method.

javax.security.jacc

PolicyContextException

Declaration

```
public class PolicyContextException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--javax.security.jacc.PolicyContextException
```

All Implemented Interfaces: `java.io.Serializable`

Description

This checked exception is thrown by implementations of the `javax.security.jacc.PolicyConfiguration` Interface, the `javax.security.jacc.PolicyConfigurationFactory` abstract class, the `javax.security.jacc.PolicyContext` utility class, and implementations of the `javax.security.jacc.PolicyContextException` Interface.

This exception is used by javax.security.jacc implementation classes to rethrow checked exceptions ocurring within an implementation that are not declared by the interface or class being implemented.

See Also: `java.lang.Exception`, `PolicyConfiguration`, `PolicyConfigurationFactory`, `PolicyContext`, `PolicyContextHandler`

Member Summary

Constructors

`PolicyContextException()`
Constructs a new PolicyContextException with `null` as its detail message.

`PolicyContextException(java.lang.String msg)`
Constructs a new PolicyContextException with the specified detail message

`PolicyContextException(java.lang.String msg, java.lang.Throwable cause)`
Constructs a new PolicyContextException with the specified detail message and cause.

`PolicyContextException(java.lang.Throwable cause)`
Constructs a new PolicyContextException with the specified cause.

Inherited Member Summary

Methods inherited from class `Object`

Inherited Member Summary
`clone(), equals(Object), finalize(), getClass(), hashCode(), notify(), notifyAll(), wait(long, int), wait(long, int), wait(long, int)`
Methods inherited from class `Throwable`
`fillInStackTrace(), getCause(), getLocalizedMessage(), getMessage(), getStackTrace(), initCause(Throwable), printStackTrace(PrintWriter), printStackTrace(PrintWriter), printStackTrace(PrintWriter), setStackTrace(StackTraceElement[]), toString()`

Constructors

PolicyContextException()

`public` **`PolicyContextException`**`()`

Constructs a new PolicyContextException with `null` as its detail message. describing the cause of the exception.

PolicyContextException(String)

`public` **`PolicyContextException`**`(java.lang.String msg)`

Constructs a new PolicyContextException with the specified detail message

Parameters:

`msg` - - a `String` containing a detail message describing the cause of the exception.

PolicyContextException(String, Throwable)

`public` **`PolicyContextException`**`(java.lang.String msg, java.lang.Throwable cause)`

Constructs a new PolicyContextException with the specified detail message and cause. The cause will be encapsulated in the constructed exception.

Parameters:

`msg` - - a `String containing a detail message describing the cause of the exception.`

`cause` - - the Throwable that is “causing” this exception to be constructed. A null value is permitted, and the value passed through this parameter may subsequently be retrieved by calling `getCause()` on the constructed exception.

PolicyContextException(Throwable)

`public` **`PolicyContextException`**`(java.lang.Throwable cause)`

Constructs a new PolicyContextException with the specified cause. The cause will be encapsulated in the constructed exception.

Parameters:

`cause` - - the Throwable that is “causing” this exception to be constructed. A null value is permitted, and the value passed through this parameter may subsequently be retrieved by calling `getCause()` on the constructed exception.

javax.security.jacc

PolicyContextHandler

Declaration

```
public interface PolicyContextHandler
```

Description

This interface defines the methods that must be implemented by handlers that are to be registered and activated by the `PolicyContext` class. The `PolicyContext` class provides methods for containers to register and activate container-specific `PolicyContext` handlers. `Policy` providers use the `PolicyContext` class to activate handlers to obtain (from the container) additional policy relevant context to apply in their access decisions. All handlers registered and activated via the `PolicyContext` class must implement the `PolicyContextHandler` interface.

See Also: `PolicyContext`, `PolicyContextException`

Member Summary

Methods

`java.lang.Object`	`getContext(java.lang.String key, java.lang.Object data)` This public method is used by the `PolicyContext` class to activate the handler and obtain from it the context object identified by the (case-sensitive) key.
`java.lang.String[]`	`getKeys()` This public method returns the keys identifying the context objects supported by the handler.
`boolean`	`supports(java.lang.String key)` This public method returns a boolean result indicating whether or not the handler supports the context object identified by the (case-sensitive) key value.

Methods

getContext(String, Object)

```
public java.lang.Object getContext(java.lang.String key, java.lang.Object data)
            throws PolicyContextException
```

This public method is used by the `PolicyContext` class to activate the handler and obtain from it the context object identified by the (case-sensitive) key. In addition to the key, the handler will be activated with the handler data value associated within the `PolicyContext` class with the thread on which the call to this method is made.

Note that the policy context identifier associated with a thread is available to the handler by calling PolicyContext.getContextID().

Parameters:

`key` - a String that identifies the context object to be returned by the handler. The value of this paramter must not be null.

`data` - the handler data `Object` associated with the thread on which the call to this method has been made. Note that the value passed through this parameter may be `null`.

Returns: The container and handler specific `Object` containing the desired context. A `null` value may be returned if the value of the corresponding context is null.

Throws:

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException

getKeys()

```
public java.lang.String[] getKeys()
            throws PolicyContextException
```

This public method returns the keys identifying the context objects supported by the handler. The value of each key supported by a handler must be a non-null `String` value.

Returns: an array containing `String` values identifing the context objects supported by the handler. The array must not contain duplicate key values. In the unlikely case that the Handler supports no keys, the handler must return a zero length array. The value null must never be returned by this method.

Throws:

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException

supports(String)

```
public boolean supports(java.lang.String key)
            throws PolicyContextException
```

This public method returns a boolean result indicating whether or not the handler supports the context object identified by the (case-sensitive) key value.

Parameters:

`key` - a `String` value identifying a context object that could be supported by the handler. The value of this parameter must not be null.

Returns: a boolean indicating whether or not the context object corresponding to the argument key is handled by the handler.

Throws:

`PolicyContextException` - if the implementation throws a checked exception that has not been accounted for by the method signature. The exception thrown by the implementation class will be encapsulated (during construction) in the thrown PolicyContextException

javax.security.jacc

WebResourcePermission

Declaration

```
public final class WebResourcePermission extends java.security.Permission implements
    java.io.Serializable

java.lang.Object
  |
  +--java.security.Permission
        |
        +--javax.security.jacc.WebResourcePermission
```

All Implemented Interfaces: `java.security.Guard`, `java.io.Serializable`

Description

Class for Servlet web resource permissions. A WebResourcePermission is a named permission and has actions.

The name of a WebResourcePermission (also referred to as the target name) identifies the Web resources to which the permission pertains.

Implementations of this class MAY implement newPermissionCollection or inherit its implementation from the super class.

See Also: `java.security.Permission`

Member Summary

Constructors

	`WebResourcePermission(javax.servlet.http.HttpServletRequest request)` Creates a new WebResourcePermission from the HttpServletRequest object.
	`WebResourcePermission(java.lang.String name, java.lang.String actions)` Creates a new WebResourcePermission with the specified name and actions.
	`WebResourcePermission(java.lang.String urlPatternSpec, java.lang.String HTTPMethods)` Creates a new WebResourcePermission with name corresponding to the URLPatternSpec, and actions composed from the array of HTTP methods.

Methods

`boolean`	`equals(java.lang.Object o)` Checks two WebResourcePermission objects for equality.
`java.lang.String`	`getActions()` Returns a canonical String representation of the actions of this WebResourcePermission.
`int`	`hashCode()` Returns the hash code value for this WebResourcePermission.
`boolean`	`implies(java.security.Permission permission)` Determines if the argument Permission is “implied by” this WebResourcePermission.

Inherited Member Summary

Methods inherited from class `Object`

`clone(), finalize(), getClass(), notify(), notifyAll(), wait(long, int), wait(long, int), wait(long, int)`

Methods inherited from class `Permission`

`checkGuard(Object), getName(), newPermissionCollection(), toString()`

Constructors

WebResourcePermission(HttpServletRequest)

```
public WebResourcePermission(javax.servlet.http.HttpServletRequest request)
```

Creates a new WebResourcePermission from the HttpServletRequest object.

Parameters:

`request` - the HttpServletRequest object corresponding to the Servlet operation to which the permission pertains. The permission name is the substring of the requestURI (HttpServletRequest.getRequestURI()) that begins after the contextPath (HttpServletRequest.getContextPath()). When the substring operation yields the string "/", the permission is constructed with the empty string as its name. The permission's actions field is obtained from HttpServletRequest.getMethod().

WebResourcePermission(String, String)

```
public WebResourcePermission(java.lang.String name, java.lang.String actions)
```

Creates a new WebResourcePermission with the specified name and actions.

The name contains a URLPatternSpec that identifies the web resources to which the permissions applies. The syntax of a URLPatternSpec is as follows:

```
URLPatternList ::= URLPattern | URLPatternList colon URLPattern
URLPatternSpec ::= null | URLPattern | URLPattern colon URLPatternList
```

A null URLPatternSpec is translated to the default URLPattern, "/", by the permission constructor. The empty string is an exact URLPattern, and may occur anywhere in a URLPatternSpec that an exact URLPattern may occur. The first URLPattern in a URLPatternSpec may be any of the pattern types, exact, path-prefix, extension, or default as defined in the *Java Servlet Specification)*. When a URLPatternSpec includes a URLPatternList, the patterns of the URLPatternList identify the resources to which the permission does NOT apply and depend on the pattern type and value of the first pattern as follows:

- No pattern may exist in the URLPatternList that matches the first pattern.
- If the first pattern is a path-prefix pattern, only exact patterns matched by the first pattern and path-prefix patterns matched by, but different from, the first pattern may occur in the URLPatternList.
- If the first pattern is an extension pattern, only exact patterns that are matched by the first pattern and path-prefix patterns may occur in the URLPatternList.

- If the first pattern is the default pattern, “/”, any pattern except the default pattern may occur in the URLPatternList.
- If the first pattern is an exact pattern a URLPatternList must not be present in the URLPatternSpec.

The actions parameter contains a comma seperated list of HTTP methods. The syntax of the actions parameter is defined as follows:

```
ExtensionMethod ::= any token as defined by RFC 2616
          (that is, 1*[any CHAR except CTLs or separators])
HTTPMethod ::= “GET” | “POST” | “PUT” | “DELETE” | “HEAD” |
         “OPTIONS” | “TRACE” | ExtensionMethod

HTTPMethodList ::= HTTPMethod | HTTPMethodList comma HTTPMethod

HTTPMethodExceptionList ::= exclaimationPoint HTTPMethodList

HTTPMethodSpec ::= null | HTTPMethodExceptionList |
         HTTPMethodList
```

If duplicates occur in the HTTPMethodSpec they must be eliminated by the permission constructor.

A null or empty string HTTPMethodSpec indicates that the permission applies to all HTTP methods at the resources identified by the URL pattern.

If the HTTPMethodSpec contains an HTTPMethodExceptionList (i.e., it begins with an exclaimationPoint), the permission pertains to all methods except those occuring in the exception list.

Parameters:

`name` - the URLPatternSpec that identifies the application specific web resources to which the permission pertains. All URLPatterns in the URLPatternSpec are relative to the context path of the deployed web application module, and the same URLPattern must not occur more than once in a URLPatternSpec. A null URLPatternSpec is translated to the default URLPattern, “/”, by the permission constructor.

`actions` - identifies the HTTP methods to which the permission pertains. If the value passed through this parameter is null or the empty string, then the permission pertains to all the possible HTTP methods.

WebResourcePermission(String, String[])

```
public WebResourcePermission(java.lang.String urlPatternSpec,
            java.lang.String[] HTTPMethods)
```

Creates a new WebResourcePermission with name corresponding to the URLPatternSpec, and actions composed from the array of HTTP methods.

Parameters:

`urlPatternSpec` - the URLPatternSpec that identifies the application specific web resources to which the permission pertains. All URLPatterns in the URLPatternSpec are relative to the context path of the deployed web application module, and the same URLPattern must not occur more than once in a URLPatternSpec. A null URLPatternSpec is translated to the default URLPattern, “/”, by the permission constructor.

`HTTPMethods` - an array of strings each element of which contains the value of an HTTP method. If the value passed through this parameter is null or is an array with no elements, then the permission pertains to all the possible HTTP methods.

Methods

equals(Object)

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

Checks two WebResourcePermission objects for equality. WebResourcePermission objects are equivalent if their URLPatternSpec and (canonicalized) actions values are equivalent. The URLPatternSpec of a reference permission is equivalent to that of an argument permission if their first patterns are equivalent, and the patterns of the URLPatternList of the reference permission collectively match exactly the same set of patterns as are matched by the patterns of the URLPatternList of the argument permission.

Two Permission objects, P1 and P2, are equivalent if and only if P1.implies(P2) && P2.implies(P1).

Overrides: `equals` in class `Permission`

Parameters:
`o` - the WebResourcePermission object being tested for equality with this WebResourcePermission.

Returns: true if the argument WebResourcePermission object is equivalent to this WebResourcePermission.

getActions()

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

Returns a canonical String representation of the actions of this WebResourcePermission. In the canonical form, predefined methods preceed extension methods, and within each method classification the corresponding methods occur in ascending lexical order. There may be no duplicate HTTP methods in the canonical form, and the canonical form of the set of all HTTP methods is the value null.

Overrides: `getActions` in class `Permission`

Returns: a String containing the canonicalized actions of this WebResourcePermission (or the null value).

hashCode()

```
public int hashCode()
```

Returns the hash code value for this WebResourcePermission. The properties of the returned hash code must be as follows:

- During the lifetime of a Java application, the hashCode method must return the same integer value, every time it is called on a WebResourcePermission object. The value returned by hashCode for a particular WebResourcePermission need not remain consistent from one execution of an application to another.
- If two WebResourcePermission objects are equal according to the equals method, then calling the hashCode method on each of the two Permission objects must produce the same integer result (within an application).

Overrides: `hashCode` in class `Permission`

Returns: the integer hash code value for this object.

implies(Permission)

```
public boolean implies(java.security.Permission permission)
```

Determines if the argument Permission is “implied by” this WebResourcePermission. For this to be the case, all of the following must be true:

- The argument is an instanceof WebResourcePermission
- The first URLPattern in the name of the argument permission is matched by the first URLPattern in the name of this permission.
- The first URLPattern in the name of the argument permission is NOT matched by any URLPattern in the URLPatternList of the URLPatternSpec of this permission.
- If the first URLPattern in the name of the argument permission matches the first URLPattern in the URLPatternSpec of this permission, then every URLPattern in the URLPatternList of the URLPatternSpec of this permission is matched by a URLPattern in the URLPatternList of the argument permission.
- The HTTP methods represented by the actions of the argument permission are a subset of the HTTP methods represented by the actions of this permission.

URLPattern matching is performed using the *Servlet matching rules* where two URL patterns match if they are related as follows:

- their pattern values are String equivalent, or
- this pattern is the path-prefix pattern “/*”, or
- this pattern is a path-prefix pattern (that is, it starts with “/” and ends with “/*”) and the argument pattern starts with the substring of this pattern, minus its last 2 characters, and the next character of the argument pattern, if there is one, is “/”, or
- this pattern is an extension pattern (that is, it starts with “*.”) and the argument pattern ends with this pattern, or
- the reference pattern is the special default pattern, “/”, which matches all argument patterns.

All of the comparisons described above are case sensitive.

Overrides: `implies` in class `Permission`

Parameters:

`permission` - “this” WebResourcePermission is checked to see if it implies the argument permission.

Returns: true if the specified permission is implied by this object, false if not.

javax.security.jacc

WebRoleRefPermission

Declaration

```
public final class WebRoleRefPermission extends java.security.Permission implements
    java.io.Serializable

java.lang.Object
  |
  +--java.security.Permission
        |
        +--javax.security.jacc.WebRoleRefPermission
```

All Implemented Interfaces: `java.security.Guard`, `java.io.Serializable`

Description

Class for Servlet *isUserInRole (String reference)* permissions. A WebRoleRefPermission is a named permission and has actions.

The name of an WebRoleRefPermission (also referred to as the target name) identifies a Web resource by the servlet name (in the deployment descriptor corresponding to the component from which the call to *isUserInRole (String reference)* is being made.

The actions of an WebRoleRefPermission identifies the role reference to which the permission applies. A WebRoleRefPermission is checked to determine if the subject is a member of the role identified by the reference.

Implementations of this class MAY implement newPermissionCollection or inherit its implementation from the super class.

See Also: `java.security.Permission`

Member Summary

Constructors

	`WebRoleRefPermission(java.lang.String name, java.lang.String actions)` Creates a new WebRoleRefPermission with the specified name and actions.

Methods

`boolean`	`equals(java.lang.Object o)` Checks two WebRoleRefPermission objects for equality.
`java.lang.String`	`getActions()` Returns a canonical String representation of the actions of this WebRoleRef-Permission.
`int`	`hashCode()` Returns the hash code value for this WebRoleRefPermission.
`boolean`	`implies(java.security.Permission permission)` Determines if the argument Permission is “implied by” this WebRoleRefPermission.

Inherited Member Summary

Methods inherited from class `Object`

`clone(), finalize(), getClass(), notify(), notifyAll(), wait(long, int), wait(long, int), wait(long, int)`

Methods inherited from class `Permission`

`checkGuard(Object), getName(), newPermissionCollection(), toString()`

Constructors

WebRoleRefPermission(String, String)

`public` **`WebRoleRefPermission`**`(java.lang.String name, java.lang.String actions)`

Creates a new WebRoleRefPermission with the specified name and actions.

Parameters:

`name` - the servlet-name that identifies the application specific web resource in whose context the role references are to be evaluated.

`actions` - identifies the role reference to which the permission pertains. The role reference is scoped to the Web resource identified in the name parameter. The value of the role reference must not be `null` or the empty string.

Methods

equals(Object)

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

Checks two WebRoleRefPermission objects for equality. WebRoleRefPermission objects are equivalent if they have case equivalent name and actions values.

Two Permission objects, P1 and P2, are equivalent if and only if P1.implies(P2) && P2.implies(P1).

The name and actions comparisons described above are case sensitive.

Overrides: `equals` in class `Permission`

Parameters:

`o` - the WebRoleRefPermission object being tested for equality with this WebRoleRefPermission.

Returns: true if the argument WebRoleRefPermission object is equivalent to this WebRoleRefPermission.

getActions()

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

Returns a canonical String representation of the actions of this WebRoleRefPermission.

Overrides: `getActions` in class `Permission`

Returns: a String containing the canonicalized actions of this WebRoleRefPermission.

hashCode()

```
public int hashCode()
```

Returns the hash code value for this WebRoleRefPermission. The properties of the returned hash code must be as follows:

- During the lifetime of a Java application, the hashCode method must return the same integer value, every time it is called on a WebRoleRefPermission object. The value returned by hashCode for a particular WebRoleRefPermission need not remain consistent from one execution of an application to another.
- If two WebRoleRefPermission objects are equal according to the equals method, then calling the hashCode method on each of the two Permission objects must produce the same integer result (within an application).

Overrides: `hashCode` in class `Permission`

Returns: the integer hash code value for this object.

implies(Permission)

```
public boolean implies(java.security.Permission permission)
```

Determines if the argument Permission is "implied by" this WebRoleRefPermission. For this to be the case,

- The argument must be an instanceof WebRoleRefPermission
- with name equivalent to this WebRoleRefPermission, and
- with role reference equivalent to this WebRoleRefPermission (as defined in their actions).

The comparisons described above are case sensitive.

Overrides: `implies` in class `Permission`

Parameters:

`permission` - "this" WebRoleRefPermission is checked to see if it implies the argument permission.

Returns: true if the specified permission is implied by this object, false if not.

javax.security.jacc

WebUserDataPermission

Declaration

```
public final class WebUserDataPermission extends java.security.Permission implements
    java.io.Serializable

java.lang.Object
  |
  +--java.security.Permission
        |
        +--javax.security.jacc.WebUserDataPermission
```

All Implemented Interfaces: `java.security.Guard`, `java.io.Serializable`

Description

Class for Servlet Web user data permissions. A WebUserDataPermission is a named permission and has actions.

The name of a WebUserDataPermission (also referred to as the target name) identifies a Web resource by its context path relative URL pattern.

See Also: `java.security.Permission`

Member Summary

Constructors

`WebUserDataPermission(javax.servlet.http.HttpServletRequest request)`
Creates a new WebUserDataPermission from the HttpServletRequest object.

`WebUserDataPermission(java.lang.String name, java.lang.String actions)`
Creates a new WebUserDataPermission with the specified name and actions.

`WebUserDataPermission(java.lang.String urlPatternSpec, java.lang.String HTTPMethods, java.lang.String transportType)`
Creates a new WebUserDataPermission with name corresponding to the URLPatternSpec, and actions composed from the array of HTTP methods and the transport type.

Methods

boolean	`equals(java.lang.Object o)` Checks two WebUserDataPermission objects for equality.
java.lang.String	`getActions()` Returns a canonical String representation of the actions of this WebUserDataPermission.
int	`hashCode()` Returns the hash code value for this WebUserDataPermission.
boolean	`implies(java.security.Permission permission)` Determines if the argument Permission is “implied by” this WebUserDataPermission.

Inherited Member Summary

Methods inherited from class `Object`
`clone(), finalize(), getClass(), notify(), notifyAll(), wait(long, int), wait(long, int), wait(long, int)`
Methods inherited from class `Permission`
`checkGuard(Object), getName(), newPermissionCollection(), toString()`

Constructors

WebUserDataPermission(HttpServletRequest)

`public` **`WebUserDataPermission`**`(javax.servlet.http.HttpServletRequest request)`

Creates a new WebUserDataPermission from the HttpServletRequest object.

Parameters:

`request` - the HttpServletRequest object corresponding to the Servlet operation to which the permission pertains. The permission name is the substring of the requestURI (HttpServletRequest.getRequestURI()) that begins after the contextPath (HttpServletRequest.getContextPath()). When the substring operation yields the string "/", the permission is constructed with the empty string as its name. The HTTP method component of the permission's actions is as obtained from HttpServletRequest.getMethod(). The TransportType component of the permission's actions is determined by calling HttpServletRequest.isSecure().

WebUserDataPermission(String, String)

`public` **`WebUserDataPermission`**`(java.lang.String name, java.lang.String actions)`

Creates a new WebUserDataPermission with the specified name and actions.

The name contains a URLPatternSpec that identifies the web resources to which the permissions applies. The syntax of a URLPatternSpec is as follows:

```
URLPatternList ::= URLPattern | URLPatternList colon URLPattern
URLPatternSpec ::= null | URLPattern | URLPattern colon URLPatternList
```

A null URLPatternSpec is translated to the default URLPattern, "/", by the permission constructor. The empty string is an exact URLPattern, and may occur anywhere in a URLPatternSpec that an exact URLPattern may occur. The first URLPattern in a URLPatternSpec may be any of the pattern types, exact, path-prefix, extension, or default as defined in the *Java Servlet Specification)*. When a URLPatternSpec includes a URLPatternList, the patterns of the URLPatternList identify the resources to which the permission does NOT apply and depend on the pattern type and value of the first pattern as follows:

- No pattern may exist in the URLPatternList that matches the first pattern.
- If the first pattern is a path-prefix pattern, only exact patterns matched by the first pattern and path-prefix patterns matched by, but different from, the first pattern may occur in the URLPatternList.
- If the first pattern is an extension pattern, only exact patterns that are matched by the first pattern and path-prefix patterns may occur in the URLPatternList.
- If the first pattern is the default pattern, "/", any pattern except the default pattern may occur in the URLPatternList.

- If the first pattern is an exact pattern a URLPatternList must not be present in the URLPatternSpec.

The actions parameter contains a comma separated list of HTTP methods that may be followed by a transportType separated from the HTTP method by a colon.

```
ExtensionMethod ::= any token as defined by RFC 2616
        (that is, 1*[any CHAR except CTLs or separators])
HTTPMethod ::= "Get" | "POST" | "PUT" | "DELETE" | "HEAD" |
        "OPTIONS" | "TRACE" | ExtensionMethod
HTTPMethodList ::= HTTPMethod | HTTPMethodList comma HTTPMethod
HTTPMethodExceptionList ::= exclaimationPoint HTTPMethodList
HTTPMethodSpec ::= emptyString | HTTPMethodExceptionList |
        HTTPMethodList
transportType ::= "INTEGRAL" | "CONFIDENTIAL" | "NONE"
actions ::= null | HTTPMethodSpec |
        HTTPMethodSpec colon transportType
```

If duplicates occur in the HTTPMethodSpec they must be eliminated by the permission constructor.

An empty string HTTPMethodSpec is a shorthand for a List containing all the possible HTTP methods.

If the HTTPMethodSpec contains an HTTPMethodExceptionList (i.e., it begins with an exclaimation-Point), the permission pertains to all methods except those occuring in the exception list.

An actions string without a transportType is a shorthand for a actions string with the value "NONE" as its TransportType.

A granted permission representing a transportType of "NONE", indicates that the associated resources may be accessed using any conection type.

Parameters:

`name` - the URLPatternSpec that identifies the application specific web resources to which the permission pertains. All URLPatterns in the URLPatternSpec are relative to the context path of the deployed web application module, and the same URLPattern must not occur more than once in a URLPatternSpec. A null URLPatternSpec is translated to the default URLPattern, "/", by the permission constructor.

`actions` - identifies the HTTP methods and transport type to which the permission pertains. If the value passed through this parameter is null or the empty string, then the permission is constructed with actions corresponding to all the possible HTTP methods and transportType "NONE".

WebUserDataPermission(String, String[], String)

```
public WebUserDataPermission(java.lang.String urlPatternSpec,
            java.lang.String[] HTTPMethods, java.lang.String transportType)
```

Creates a new WebUserDataPermission with name corresponding to the URLPatternSpec, and actions composed from the array of HTTP methods and the transport type.

Parameters:

`urlPatternSpec` - the URLPatternSpec that identifies the application specific web resources to which the permission pertains. All URLPatterns in the URLPatternSpec are relative to the context path of the deployed web application module, and the same URLPattern must not occur more than once in a URLPatternSpec. A null URLPatternSpec is translated to the default URLPattern, "/", by the permission constructor.

`HTTPMethods` - an array of strings each element of which contains the value of an HTTP method. If the value passed through this parameter is null or is an array with no elements, then the permission is constructed with actions corresponding to all the possible HTTP methods.

`transportType` - a String whose value is a transportType. If the value passed through this parameter is null, then the permission is constructed with actions corresponding to transportType "NONE".

Methods

equals(Object)

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

Checks two WebUserDataPermission objects for equality. WebUserDataPermission objects are equivalent if their URLPatternSpec and (canonicalized) actions values are equivalent. The URLPatternSpec of a reference permission is equivalent to that of an argument permission if their first patterns are equivalent, and the patterns of the URLPatternList of the reference permission collectively match exactly the same set of patterns as are matched by the patterns of the URLPatternList of the argument permission.

Two Permission objects, P1 and P2, are equivalent if and only if P1.implies(P2) && P2.implies(P1).

Overrides: `equals` in class `Permission`

Parameters:

`o` - the WebUserDataPermission object being tested for equality with this WebUserDataPermission.

Returns: true if the argument WebUserDataPermission object is equivalent to this WebUserDataPermission.

getActions()

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

Returns a canonical String representation of the actions of this WebUserDataPermission. The canonical form of the actions of a WebUserDataPermission is described by the following syntax description.

```
ExtensionMethod ::= any token as defined by RFC 2616
          (that is, 1*[any CHAR except CTLs or separators])
HTTPMethod ::= "GET" | "POST" | "PUT" | "DELETE" | "HEAD" |
          "OPTIONS" | "TRACE" | ExtensionMethod

HTTPMethodList ::= HTTPMethod | HTTPMethodList comma HTTPMethod
HTTPMethodExceptionList ::= exclaimationPoint HTTPMethodList
HTTPMethodSpec ::= emptyString | HTTPMethodExceptionList |
         HTTPMethodList
transportType ::= "INTEGRAL" | "CONFIDENTIAL" | "NONE"
actions ::= null | HTTPMethodList |
         HTTPMethodSpec colon transportType
```

If the permission's HTTP methods correspond to the entire HTTP method set and the permission's transport type is "INTEGRAL" or "CONFIDENTIAL", the HTTP methods shall be represented in the canonical form by an emptyString HTTPMethodSpec. If the permission's HTTP methods correspond to the entire HTTP method set, and the permission's transport type is not "INTEGRAL"or "CONFIDENTIAL", the canonical actions value shall be the null value.

If the permission's methods do not correspond to the entire HTTP method set, duplicates must be eliminated and the remaining elements must be ordered such that the predefined methods preceed the extension methods, and such that within each method classification the corresponding methods occur in ascending lexical order. The resulting (non-emptyString) HTTPMethodSpec must be included in the canonical form, and if the permission's transport type is not "INTEGRAL" or "CONFIDENTIAL", the canonical actions value must be exactly the resulting HTTPMethodSpec.

Overrides: `getActions` in class `Permission`

Returns: a String containing the canonicalized actions of this WebUserDataPermission (or the null value).

hashCode()

```
public int hashCode()
```

Returns the hash code value for this WebUserDataPermission. The properties of the returned hash code must be as follows:

- During the lifetime of a Java application, the hashCode method shall return the same integer value every time it is called on a WebUserDataPermission object. The value returned by hashCode for a particular EJBMethod permission need not remain consistent from one execution of an application to another.
- If two WebUserDataPermission objects are equal according to the equals method, then calling the hashCode method on each of the two Permission objects must produce the same integer result (within an application).

Overrides: `hashCode` in class `Permission`

Returns: the integer hash code value for this object.

implies(Permission)

```
public boolean implies(java.security.Permission permission)
```

Determines if the argument Permission is "implied by" this WebUserDataPermission. For this to be the case all of the following must be true:

- The argument is an instanceof WebUserDataPermission.
- The first URLPattern in the name of the argument permission is matched by the first URLPattern in the name of this permission.
- The first URLPattern in the name of the argument permission is NOT matched by any URLPattern in the URLPatternList of the URLPatternSpec of this permission.
- If the first URLPattern in the name of the argument permission matches the first URLPattern in the URLPatternSpec of this permission, then every URLPattern in the URLPatternList of the URLPatternSpec of this permission is matched by a URLPattern in the URLPatternList of the argument permission.
- The HTTP methods represented by the actions of the argument permission are a subset of the HTTP methods represented by the actions of this permission.
- The transportType in the actions of this permission either corresponds to the value "NONE", or equals the transportType in the actions of the argument permission.

URLPattern matching is performed using the *Servlet matching rules* where two URL patterns match if they are related as follows:

- their pattern values are String equivalent, or
- this pattern is the path-prefix pattern "/*", or
- this pattern is a path-prefix pattern (that is, it starts with "/" and ends with "/*") and the argument pattern starts with the substring of this pattern, minus its last 2 characters, and the next character of the argument pattern, if there is one, is "/", or
- this pattern is an extension pattern (that is, it starts with "*.") and the argument pattern ends with this

pattern, or

- the reference pattern is the special default pattern, "/", which matches all argument patterns.

All of the comparisons described above are case sensitive.

Overrides: `implies` in class `Permission`

Parameters:

`permission` - "this" WebUserDataPermission is checked to see if it implies the argument permission.

Returns: true if the specified permission is implied by this object, false if not.

APPENDIX A

Related Documents

This specification refers to the following documents. The terms used to refer to the documents in this specification are included in brackets.

S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, [Keywords] Harvard University, March 1997

Java™ 2 Platform, Enterprise Edition Specification Version 1.4 [Java EE specification] Copyright 1999-2003, Sun Microsystems, Inc. Available at http://java.sun.com/j2ee/docs.html.

Java™ 2 Platform, Standard Edition, v1.4.0 API Specification [Java SE specification]. Copyright 1993-2003, Sun Microsystems, Inc. Available at http://java.sun.com/j2se/1.4/docs/api/index.html.

Enterprise JavaBeans™ Specification, Version 2.1 [EJB specification]. Copyright 1998-2003, Sun Microsystems, Inc. Available at http://java.sun.com/products/ejb.

Java™ Servlet Specification, Version 2.4 [Servlet specification]. Copyright 1998-2003, Sun Microsystems, Inc. Available at http://java.sun.com/products/servlet.

Java™ Authentication and Authorization Service (JAAS) 1.0 [JAAS specification]. Copyright 1999-2003, Sun Microsystems, Inc. Available at http://java.sun.com/products/jaas.

APPENDIX B

Issues

The following sections document the more noteworthy issues that have been discussed by the Expert Group. These sections are included in the Final Release as they provide insight into the discussions and decisions which shaped the form of the current specification. All of these issues have been resolved, and their resolutions are described below and reflected in the document.

B.1 Configuration Context and Policy Context Identifiers

The PolicyConfiguration interface associates Configuration Context Identifiers with policy statements, which themselves contain embedded policy context identifiers. There needs to be more explanation of the purpose and use models of these context identifiers. Configuration context identifiers should only be assigned by the Provider, to eliminate problems with ambiguity. In the case of createRole, we allow a configuration context identifier to be passed to createRole so that one configuration context id can be used for all of the roles in an application/module. It would probably be a good idea to allow the unchecked and excluded policy collections to also share the same context id. This would reduce the complexity of the identity mapping, but it would make it harder for the provider to ensure uniqueness of identifiers. We also want to support deployment and undeployment of modules, within a multi-module policy configuration context. That is, the modules share the same roles, but their individual policy statements are differentiated by policy context id within these roles.

Note – Regarding policy context identifiers, it will not be possible to surgically replace the policy statements corresponding to a module, if modules within a policy configuration context share the same policy context identifiers.

Resolution– The PolicyConfiguration interface has been redesigned to support both the factory and finder patterns, and to include an inService method to allow a container to check if a PolicyConfiguration with a given identifier already exists. The interface also includes the concept of linked PolicyConfigurations to identify those PolicyConfigurations that must share the same principal-to-role mappings.

B.2 Configuration of Permissions with Parameters

The PolicyConfiguration interface is used to communicate policy statements to Policy. An element of these statements is a PermissionCollection that may contain EJBMethodPermission and EJBRoleRefPermission objects that may have been constructed with embedded references to argument arrays or EntityBean instances. The contract must state whether such permissions may be passed through the PolicyConfiguration interface, and what the responsibility of the provider shall be should it occur.

Resolution– resolved via the introduction of PolicyContext handlers and the corresponding removal of the ability to include such information in permission constrictions.

B.3 Extensibility of the PolicyConfiguration Interface

For example, the PolicyConfiguration interface does not include methods that may be used to interrogate Policy to determine the list of configured policy configuration contexts. We should also consider whether the interface should be extended to support the configuration of additional forms (other than unchecked, and excluded) logical policy statements

Resolution– (Partial) with the change to finder semantics, that is getPolicyConfiguration, and the addition of the inService method to PolicyConfigurationFactory.

B.4 Directory Scoped Extension matching patterns

Resolution– We will not require that they be supported by policy providers, nor will we require that policy providers reject other than the patterns defined by servlet.

B.5 Evolution of Deployment Policy Language

The PolicyConfiguration and Policy Decision Subcontracts should be generalized to sustain evolution in the declarative authorization policy representations used in deployment descriptors. One dimension of this evolution, would be a change from DTDs to schema.

Resolution– Some generalization in the PolicyConfiguration interface has occurred as a result of the removal of policy context identifiers from permissions such that any permission objects may be configured through this interface.

B.6 Principals Passed to Providers in Subjects

The provider is expected to do principal-to-role mapping, but we have not allowed the provider to assume that it is working with a companion authentication module. We have also not defined standard principals for containers to put in the subjects used when they ask Policy to make decisions for them. So, it is unclear how providers will be able to do Principal-to-Role mapping.

Resolution– We decoupled consideration of this issue from notions of principal selection imposed by the getCallerPrincipal and getUserPrincipal methods of EJB and Servlet respectively. We clarified that all principals in an AccessControlContext shall be available to the policy module for use in principal to role mapping. We added a requirement with respect to asserting or vouching authorities to ensure that principals corresponding to authorities are not misinterpreted by providers as principals of the subject (see Section 4.8, “Checking the Caller for a Permission"). Moreover, we concluded that independent of this contract, a policy module must be familiar with the principals

(i.e. security attributes) assigned to subjects as the result of authentication in its operational environment and for which it must evaluate policy.

B.7 Clarification of Servlet Constraint Matching Semantics

The definition of the security-constraint matching and enforcement semantics are under specified in the Servlet 2.2 and 2.3 specifications. The contract defined in this document has clarified these semantics; however there will be an issue until these clarifications are incorporated in Servlet.

Resolution– Servlet 2.4 specification will include a more complete description of the processing of constraints.

B.8 References and Arguments in EJBMethodPermisison

When a container constructs an EJBMethodPermission as part of its policy decision subcontract, it may include a reference to the EJB (for an EntityBean) and the arguments to the method in the constructed permission. Inclusion of this additional context by containers is optional for performance reasons, yet it has been suggested that the contract provide a way (perhaps via a callback or an exception thrown by the provider) for the container to find out whether or not such information would be used by the provider.

Resolution– Resolved with introduction of policy context handlers

B.9 Permission Spanning in RoleRefPermission

The EJBRoleRefPermission and WebRoleRefPermission objects support the checking of multiple “references” in a single permission check. This functionality was motivated by recurring requests to extend the Java EE “inRole” APIs to allow multiple role references to be evaluated in a single call. The permission classes noted above, currently support this functionality, at the cost of having to span permissions in collection implication. The most direct consequence of this spanning is that the new Permission Collection methods of these Permission classes must not

return null, as they must return a PermissionCollection capable of doing the permission specific spanning.

Resolution– The replacement paradigm has been changed such that it should no longer be possible for providers to depend on custom implementations of the permission classes defined by this specification. Accordingly, the complexity introduced by spanning should be attenuated in the reference implementation.

B.10 Integrating Principal-to-Role Mapping with the Deployer Console

The desire for a single user interface for security administration works somewhat counter to Java EE platforms that provide their own administrative console for tasks like principal-to-role mapping. That said, we need to preserve existing platform administrative consoles by defining a method to integrate provider specific principal-to-role mapping dialogs via the existing platform consoles. We must ensure alignment with JSR 88, which is solving a slightly different but overlapping problem. JSR 88 is defining how a platform with fixed deployment interfaces can be integrated with an administration console provided by another party.

Resolution– This is optional functionality. Providers may choose to provide a JSR 88 config bean that encapsulates a provider specific principal-to-role management interface. Containers that wish to have tight integration between their deployment consoles and a provider supplied principal-to-role management interface, should pass the provider supplied config bean among the config beans they are required by JSR 88 to return to a deployment tool.

B.11 PolicyContext Identifiers are Unknown to Components

Although not strictly speaking within the scope of this JSR, the work of this JSR empowers application components to use the Java SE policy decision interface to perform their own access control decisions. The permissions defined by this specification must be constructed with an embedded policy context identifier so that the policy provider can evaluate the permission in the proper deployment context (i.e policy configuration). As currently defined, the specification does not provide a component with access to its policy configuration identifiers, and as such a

component can not check any permissions which implement the PolicyContext interface.

Resolution– resolved by moving policy context identifiers out of the permissions, into the PolicyContext utility class

---

B.12 JAAS Policy Interface expects Providers to be able to getPermissions

Not all Policy providers can, or find it convenient or efficient, to determine all of the permissions granted to an access control context. The JAAS Policy decision interface, and the use of this interface by the JAAS SubjectDomainCombiner, impede the integration of Policy Providers that are unable to enumerate all the permissions that pertain to a subject/protection domain before returning from Policy.getPermissions().

Resolution– Added requirement to Provider Configuration Subcontract, that the `javax.security.auth.SubjectDomainCombiner` of a J2EE 1.3 application server must `combine` into the permission collection returned by `javax.security.auth.Policy.getPermisions`. Recommended same of J2EE 1.4 application servers.

B.13 Implementing Web Security Constraints as Permission

Specification of the WebResourcePermission and WebUserDataPermission classes with simple, single URL pattern names is a bad fit for the Java SE Policy decision interface. The implementation of getPermissions presents a major challenge, as the constraint model would force the implementation to preserve ungranted constraining permissions in the returned PermissionCollection. It also would not be possible to implement the enumeration functionality available through the elements method of the collection. Perhaps more significant, the mapping of security constraints to simple, single URL pattern names would require a special more complex Policy provider rule combining algorithm, and as such, would render the default Java Policy provider incompetent to process such permissions. The last point is in direct conflict with a stated goal of the specification.

Resolution– The translation of web security constraints into Java SE permissions was modified such that the URL pattern names of the WebResource and WebUserData permissions include a representation of the URL patterns to which the permission does NOTapply. The permission implies logic was enhanced to take this change into account. As a result of these changes these permissions may be processed by the default Java SE Policy module like any other Java SE permission.

B.14 Exception Handling

The first PFD did not define error handling for the methods of the PolicyConfigurationFactory and PolicyContext classes, or for the PolicyConfiguration and PolicyContextHandler interfaces. Also, no provision was provided for implementation classes to pass checked exceptions out through the defined interfaces and classes.

Resolution– A PolicyContextException class was added to the javax.security.jacc package, and the methods of the classes and interfaces identified above were modified to throw this checked exception as appropriate.

B.15 PolicyConfiguration Commit

The first PFD did not provide a way for container deployment tools to indicate when the translation of a policy context was complete and available for assimilation into the associated Policy provider. It had been assumed that the Policy.refresh method could serve this purpose, until it was discovered that depending on Policy.refresh for this purpose would preclude parallelism in the deployment of applications.

Resolution– Added "commit" and “inService” methods to the PolicyConfiguration interface, and formalized a 3 state (i.e. open, inService, and deleted) life cycle for policy contexts. Required that the commit method be called on a PolicyConfiguration object after all of its policy statements have been added, and after it is "linked to any other module with which it must share the same

principal-to-role mapping". Also required that Policy.refresh only assimilate policy contexts in the “inService” state.

B.16 Support for ServiceEndpoint methodInterface

The definition of the EJBMethodPermission class in the first PFD did not support “ServiceEndpoint" as a valid methodInterface value. The ServiceEndpoint methodInterface was introduced by EJB 2.1.

Resolution– Added “ServiceEndpoint” as another possible value for the methodInterface component of an EJBMethodPermission methodNameSpec.

B.17 TypeNames of EJBMethodPermission Array Parameters

The syntax or syntaxes that may be used to specify array parameters are not defined by the constructors of the EJBMethodPermission class. The corresponding canonical form of such params as returned by getActions must also be specified.

Resolution– Added requirement that Array parameters be specified as ComponentType[] as apposed to in the form returned by Class.getName() (i.e. [LComponentType;).

B.18 Checking Permission on the root of a Web Application

The URLPattern, "/", cannot be used to check a permission, as it is a synonym for asking if permission to access the entire application has been granted.

Resolution– Require that the empty string be used as a replacement for "/", during the permission evaluation. Clarify the WebResourcePermission and WebUserDataPermission definitions to account for the use of the empty-string as a legitimate URLPattern in such permissions.

B.19 Calling isUserInRole from JSP not mapped to a Servlet

Checking a WebRoleRefPermission requires the name of a Servlet to identify the scope of the reference to role translation. The name of a scoping servlet has not been established for an unmapped JSP.

Resolution– For every security role in the web application add a WebRoleRefPermission to the corresponding role. The name of all such permissions shall be the empty string, and the actions of each permission shall be the corresponding role name. When checking a WebRoleRefPermission from a Web resource not mapped to a servlet, use a permission with the empty string as its name and with the argument to isUserInRole as its actions.

B.20 Support for HTTP Extension Methods

Servet 2.5 added support for HTTP extension methods (as defined in IETF RFC 2616 "Hypertext Transfer Protocol -- HTTP/1.1") to security-constraints. Support for extension methods requires changes to the WebResourcePermission and WebUserDataPermission classes and to the translation of servlet security-constraints. In general support for HTTP extension methods requires an ability to represent non-enumerable HTTP method sets in the HTTPMethodSpec components of WebResourcePermission and WebUserDataPermission actions values.

Resolution– Modified the HTTPMethodSpec constructs of WebResourcePermission and WebUserDataPermission to support an HTTPMethodExceptionList as a third form of HTTPMethodSpec. This resolution is known to have the following consequences with respect to backward compatibility:1) A permission constructed with an HTTPMethodSpec composed of an HTTPMethodList containing all the "standard" HTTP methods (i.e., "DELETE,GET,HEAD,OPTIONS,POST,PUT,TRACE) is no longer equal to and no longer implies a permission constructed with a null, empty array, or emptyString HTTPMethodSpec. 2) Permissions constructed with a null, empty array, or emptyString HTTPMethodSpec component to their actions value represent the non-enumerable (due to extension methods) set of all possible HTTP methods and are NOT equal to or implied by any permission constructed with an HTTPMethodSpec represented as an HTTPMethodList. 3) It is no longer possible to use the HTTPMethodList syntax to represent (via enumeration) the complement of a proper subset of all HTTP methods. As such, an HTTPMethodExceptionList must be used to represent any proper subset of

HTTP methods determined NOT to be constrained during the translation of servlet security-constraints. 4) The use of exception lists causes the permissions resulting from the translation of a given security-constraint configuration to differ in their actions values from those that would have been produced prior to support for HTTP extension methods. Previously translated permissions remain supported by the changed permission implementations, and (with the exceptions listed in 1 and 2 above) continue to function as they did before the change, as long as extension methods are not set in checked permissions.

B.21 Welcome File and security-constraint Processing

The relationship between welcome file processing (which can modify the effective request URI) and security-constraint processing is not defined by the Servlet Specification. Since this specification uses url-patterns derived from request URIs to name target resources in checked permissions, it is important that welcome file processing and its relationship to security-constraint processing be clearly specified. Without a clear description of this relationship, unprotected request URIs which are modified to yield effective request URIs for protected resources may inadvertently be left unprotected.

Resolution– pending Servlet clarification

B.22 Colons Within path-segment of Request URI

As defined in IETF RFC 2396 "Uniform Resource Identifiers (URI): Generic Syntax", the abs_path component of a request URI may consist of a sequence of "/" separated path segments, where the format of each segment is defined as follows:

```
segment = *pchar *( ";" param )
param   = *pchar
pchar   = unreserved | escaped |":" | "@" | "&" | "=" | "+" | "$" | ","
```

A colon character occurring within a path-seqment will be syntactically indistinguishable from colons used by the WebResourcePermission and WebUserDataPermission constructors to demarcate qualifying patterns.

Resolution– pending.

APPENDIX C

Revision History

C.1 Community Draft Version 0.3 (dated 12/13/2001)

Posted for Community Review 12/17/2001

C.2 Changes in Public Draft Version 0.1

C.2.1 General

- JCP version changed to 2.1.
- Specification title changed to J2EE™ Authorization Contract for Containers
- Added additional definitions to the terminology section for "JAAS Policy interface" and "JAAS Policy provider".
- Converted the requirements with respect to support for this specification on J2EE 1.3 and 1.4 platforms into assumptions, as any such requirements will ultimately be defined in the J2EE 1.4 Platform Specification.
- Added an Assumption and a corresponding requirement with respect to support for Policy Providers that get all permissions before returning from Policy.
- Clarified relationship to Servlet and EJB specification of authorization semantics.
- Changed all references to "VM" to "JRE"
- Changed all references to "deploy tool" to "deployment tool"

- Removed empty brackets from all method names in prose.

C.2.2 Changes to Provider Configuration Subcontract

- Rewrote replacability paradigm. New model does not require replacement of permission implementations.
- Described changes to JAAS SubjectDomainCombiner as required when contract is optionally applied in a J2EE 1.3 context.

C.2.3 Changes to Policy Configuration Subcontract

- Changed to be compatible with the changes made (for replacability) to the "Provider Configuration Subcontract".

C.2.4 Changes to Policy Decision Subcontract

- Added section describing "Component runAs Identity" to distinguish between runAs identity and caller identity. More accurately described what a container must do to set a component's runAs identity. Added requirement that container prevent component from being able to modify its runAs identity.
- Added clarification of the matching of an excluded policy statement to a granted permission.
- Clarified the policy decision algorithms in "Checking AccessControlContext Independent Grants" and "Checking the Caller for a Permission" to be compatible with the distinction between caller and runAs identity. Also factored out references to platform versions.
- In "Checking the Caller for a Permission" added requirement that caller identity not include principals of any trusted (other than the caller).
- Added new section, "Missing Policy Contexts", to make explicit the behavior of a provider when asked to check a permission in an unknown policy context.
- Rewrote the "Policy Compatibility Requirements" to indicate that a J2EE 1.4 container that uses a JAAS policy interface to perform container access decisions would not be compatible with this specification.

C.2.5 Changes to API

- Changed PolicyConfigurationFactory to be an abstract class with a static method that reads a system property to instantiate a concrete factory

implementation class. Also documented the role of PolicyConfigurationFactory in supporting the PolicyConfiguration with null context identifier.
- Specified exceptions to be thrown by methods of PolicyConfiguration interface.
- Changed the PolicyConfiguration interface to support the configuration of permissions that are not instances of PolicyContext into a PolicyConfiguration with null identifier.

C.2.6 Changes to Issues

- Resolved issue B.6,“Principals Passed to Providers in Subjects"
- Resolved issue B.9,“Permission Spanning in RoleRefPermission"
- Added new issue B.10,“Integrating Principal-to-Role Mapping with the Deployer Console"
- Added new Issue B.11,“PolicyContext Identifiers are Unknown to Components"

C.3 Changes in Public Draft Version 0.2

C.3.1 General

- Specification title changed to Java™ Authorization Contract for Containers
- Corrected audience to be the public
- In terminology: simplified definition of redeploy, corrected definition of provider, by removing permission implementations, as these will now be part of application server platform.

C.3.2 Changes to Provider Configuration Subcontract

- Replaced most references to container with JRE, as a provider integrates per JRE.

C.3.3 Changes to Policy Decision Subcontract

- In 4.4.1,“EJB Policy Decision Semantics", replace references to “subject” with “access control context”.

- In 4.5,“Component runAs Identity", softened requirement that container prevent component from modifying its runAs identity by saying that this must be the default policy.
- Changes references to “access exception” to AccessControlException.

C.3.4 Changes to Issues

- Added Introductory paragraph.
- Added new issue, B.12,“JAAS Policy Interface expects Providers to be able to getPermissions" and its resolution to ensure that this issue is documented.

C.4 Changes in Proposed Final Draft 1 Expert Draft 0.1

C.4.1 General

- The license page was changed; most notably the license number.
- Improper uses of the word “which” were replaced with the word “that”.
- The word “shall” was replaced with the word “must”.
- The 2.3 version designation was removed from references to Servlet as the applicable Servlet release is defined by the EE environment.

C.4.2 Changes to the Preface and Overview

- The preface was changed to reflect the purpose of the PFD
- the definition of hostname was modified so that hostnames are no longer required to be used in servlet policy context identifiers.
- The requirement that permissions identify the context of their use was changed to require that the context be set before permission evaluation.

C.4.3 Changes to Provider Configuration Subcontract

- Section 2.4, “PolicyContext Class and Context Handlers” was inserted to describe the PolicyContext utility class and the PolicyContextHandler interface.
- Section 2.7, “What the Application Server Must Do” was modified to include the application server’s responsibilities relating to the PolicyContext class, and to correct errors in the names of the abstract policy classes.

C.4.4 Changes to Policy Configuration Subcontract

- The examples in Section 3.1, "What a Java EE Platform's Deployment Tools Must Do", were modified to reflect changes to policy context identifiers and their removal from permission names.
- The requirement that the names of checked permissions identify the policy context was removed from Section 3.1.1, "Policy Contexts and Policy Context Identifiers"
- The linkConfiguration method name replaced the incorrect link method name in Section 3.1.1.2, "Linking Policy Contexts"
- Section 3.1.2, "Servlet Policy Context Identifiers" was moved to follow Section 3.1.1.2, "Linking Policy Contexts", and the section was made less prescriptive with respect to the format of Servlet policy context identifiers. The non-normative description of the behavior of the Tomcat server was removed.
- Section 3.1.3.1, "Translating security-constraint Elements" was made a subsection of a new Section 3.1.3, "Translating Servlet Deployment Descriptors" and changed to deal with the removal of policy context identifiers from permission names.
- The part of the translation remaining in Section 3.1.3.1, "Translating security-constraint Elements" was modified to yield an OR constraint combination semantic.
- The description of the mapping of transport guarantees to unacceptable connection types was moved to a new Section , "Mapping Transport Guarantee to Connection Type"
- Section 3.1.3.2, "Translating Servlet security-role-ref Elements" was modified to reflect the removal of policy context identifiers from permission names.
- Section 3.1.5.1, "Translating EJB method-permission Elements" was made a subsection of a new Section 3.1.5, "Translating EJB Deployment Descriptors" and changed to deal with the removal of policy context identifiers from permission names.
- A new section Section 3.1.4, "EJB Policy Context Identifiers", was added to describe the selection of EJB policy context identifiers.
- Section 3.1.5.1, "Translating EJB method-permission Elements", Section 3.1.5.2, "Translating the EJB exclude-list", and Section 3.1.5.3, "Translating EJB security-role-ref Elements" were all changed to reflect the removal of the policy context identifier from permission names.

- Section 3.1.7, "Undeploying an Application or Module" was modified to reflect the use of the PolicyContext class to define the policy context.
- Section 3.3, "Permission to Configure Policy" was changed to require that "the state of the policy statement repository" not be changed when the caller does not have the "setPolicy" permission. Also a new requirement was added that policy be configured to grant containers the "getPolicy" and "setPolicy" permissions.

C.4.5 Changes to Policy Decision Subcontract

- The name was changed to the "Policy Decision and Enforcement Contract".
- Section 4.1.2, "Evaluation of Transport Guarantees", Section 4.1.3, "Pre-dispatch Decision", and Section 4.1.4, "Application Embedded Privilege Test" were changed to reflect the removal of the policy context identifier from permission names.
- Section 4.2.1 "Servlet Constraint Matching Semantics", was replaced by two sections; Section 4.2.1, "Servlet Policy Decision Semantics", and Section 4.2.1.3, "WebResourcePermission Matching Rules". The latter describes the processing of servlet constraints in a manner related to the three types of policy statements created via the PolicyConfiguration interface.
- Section 4.2.2.1, "Servlet URL-Pattern Matching Rules" was renamed.
- Some changes were made to the last two tables of Section 4.2.2.2, "Servlet Constraint Matching Examples" to accommodate and better illustrate the OR constraint combining semantics.
- Section 4.2.3, "WebRoleRefPermission Processing Semantics" was added as the public draft mistakenly assumed that the Servlet policy model was just about constraints.
- Section 4.3.1, "EJB Pre-dispatch Decision" and Section 4.3.2, "EJB Application Embedded Privilege Test" were changed to reflect the removal of the policy context identifier from permission names.
- A new Section 4.6, "Setting the Policy Context" was added to describe how a container must set the policy context before invoking policy. This section also requires that containers be granted the setPolicy permission in all policy contexts.
- A new Section 4.6.1, "Policy Context Handlers" was added to define the requirements on containers with respect to policy context handlers. The following new sections were added to define the policy context handlers required of containers: Section 4.6.1.1, "Container Subject

Policy Context Handler", Section 4.6.1.2, "SOAPMessage Policy Context Handler", Section 4.6.1.3, "HttpServletRequest Policy Context Handler", Section 4.6.1.4, "EnterpriseBean Policy Context Handler", and Section 4.6.1.5, "EJB Arguments Policy Context Handler".

- The methods for checking policy as defined in Section 4.7, "Checking AccessControlContext Independent Grants" were reorganized such that it is clear that one of the presented alternatives must be used. Using AccessController.checkPermission was added as an additional supported alternative, and the release specific techniques were annotated as such. Also the techniques based on getPermissions were annotated as not recommended. At the end of the section a requirement was made regarding the policy context having been set prior to the evaluation.
- The same changes as described in the previous change item were applied to Section 4.8, "Checking the Caller for a Permission".
- Section 4.9, "Missing Policy Contexts" was renamed from "Unconfigured Policy Contexts" and the semantics were modified to reflect the use of the PolicyContext utility class and the designation of the null policy context id as the default.
- A new Section 4.10, "Default Policy Context" was introduced to describe requirements for chaining policy evaluation through to the provider of the default policy context.

C.4.6 Changes to API

- Replaced the PolicyContext interface with the PolicyContext class. Also changed all of the permissions such that none of them implement the PolicyContext interface and such that none of them include a policy context identifier in their names.
- Added the PolicyContextHandler interface.
- Removed the special purpose, EntityBean and Argument array constructors from the EJBMethodPermission class.
- Removed the special purpose, EntityBean constructor from the EJBRoleRefPermission class.
- Modified the actions field of the EJBRoleRefPermission and WebRoleRefPermission classes such that they contain at most a single role reference. Related to this change, also removed the newPermissionCollection method implementation from both of these classes.

- In the PolicyConfiguration interface, changed the name of the getPolicyContextId method to getContextID.
- Changed the description of the PolicyConfigurationFactory to require implementation classes to have a public no argument constructor. Also precluded the use of the null value as an argument to getPolicyConfiguration.
- Added a new constructor to the WebResourcePermission and WebUserDataPermission classes to allow an instance to be constructed from an HttpServletRequest.

C.4.7 Changes to Issues

- Changed the introductory material to indicate that all of the issues have been resolved.
- The resolution of Issue B.2,“Configuration of Permissions with Parameters", was changed to reflect the introduction of policy context handlers.
- Issue B.5,“Evolution of Deployment Policy Language", was partially resolved by removing the requirement that permissions added via the PolicyConfiguration interface have policy context identifiers in their names.
- Issue B.7,“Clarification of Servlet Constraint Matching Semantics", was resolved with the rewrite of Section 4.2.1.3,“WebResourcePermission Matching Rules", and with the expectation that the Servlet EG will adopt a change to section SRV.12.8 of the Servlet specification.
- Issue B.8,“References and Arguments in EJBMethodPermisison", was resolved with the introduction of policy context handlers.
- Issue B.10,“Integrating Principal-to-Role Mapping with the Deployer Console", was made optional functionality.
- Issue B.11,“PolicyContext Identifiers are Unknown to Components", was resolved by introducing the PolicyContext utility class.

C.5 Changes in Proposed Final Draft 1 Expert Draft 0.2

C.5.1 Changes to the Preface and Overview

- The restriction that entities be identified by principal was removed from the definition of grant.

C.5.2 Changes to Policy Configuration Subcontract

- In Section 3.1, "What a Java EE Platform's Deployment Tools Must Do", the argument to linkConfiguration was corrected in the example.

C.5.3 Changes to Policy Decision Subcontract

- Section 4.2.2.1, "Servlet URL-Pattern Matching Rules" was modified to indicate that pattern length only is significant among path prefix matches.
- A description of the content of the tables and how they should be interpreted was added to Section 4.2.2.2, "Servlet Constraint Matching Examples".
- Section 4.2.3, "WebRoleRefPermission Processing Semantics" was added as the public draft mistakenly assumed that the Servlet policy model was just about constraints.
- Section 4.3.1, "EJB Pre-dispatch Decision" and Section 4.1.4, "Application Embedded Privilege Test" were changed to reflect the removal of the policy context identifier from permission names.

C.5.4 Changes to History

- The history section was completed to reflect the changes made in Version 0.1 and 0.2

C.6 Changes in Proposed Final Draft 1 Expert Draft 0.3

C.6.1 Changes to the Preface and Overview

- The requirement that applicable constraints be selected by best-match was rephrased to define best-match as it is defined in this spec and the Servlet specification.

C.6.2 Changes to Policy Configuration Subcontract

- A clarifying sentence was added to the end of Section 3.2, "What the Provider Must Do" to make it clear that this specification does not prescribe the policy language or the methods used within providers to implement the defined policy and role requirements.

C.6.3 Changes to Policy Decision Subcontract

- Section 4.2.3, “WebRoleRefPermission Processing Semantics” was simplified, as much of its content was not pertinent to the WebRoleRefPermission class.
- Section 4.4.2, “EJB Permission Matching Rules” was changed to reflect the change to a single role in the actions of the EJBRoleRefPermission class.
- In Section 4.6.1.1, “Container Subject Policy Context Handler”, the key for the "Subject Policy Context Handler" was changed to javx.security.auth.Subject.container, and the semantics were modified to return the caller or runAs identity as appropriate.
- In Section 4.6.1.4, “EnterpriseBean Policy Context Handler”, the handler return type was corrected.

C.6.4 Changes to API

- The resolution of the class diagram was improved by changing to a black and white image.

C.7 Changes in Proposed Final Draft 2 Expert Draft 1

C.7.1 General

- In many places through out the document, replaced used of the phrase “policy configuration” with “policy context”, and adopted the practice of using PolicyConfiguration to refer to the configuration interface of a policy context.

C.7.2 Changes to Preface

- Updated Status section
- Acknowledged all contributors, including RI and TCK team, and all those who commented on the specification.

C.7.3 Changes to Overview

- Added dashed lines to Figure 1.1to represent PolicyContext interactions.
- Modified requirement 7, to reflect change is treatment of permissions derived from security-constraints.

C.7.4 Changes to Provider Configuration Subcontract

- Added two new sentences to the end of Section 2.1, "Policy Implementation Class", to make it clear that this contract is dependent on the standard Java Policy replacement mechanisms, and to make it clear that containers must support replacability.
- In Section 2.7, "What the Application Server Must Do", added all elements of the jacc package to the list of things that an application server must bundle.
- In Section 2.7, "What the Application Server Must Do", the requirement for javax.security.auth.Policy replacement was softened such that it only applies to 1.3 application servers that choose to support this specification.
- In Section 2.7, "What the Application Server Must Do", reintroduced the requirement that setPolicy not be called again, to ensure more than temporary Policy replacement.

C.7.5 Changes to Policy Configuration Subcontract

- In the examples in Section 3.1, "What a Java EE Platform's Deployment Tools Must Do", the type of the declared permission was corrected to agree with constructed type, and "petID" was changed to "petContextID" (as a clarification).
- In the examples in Section 3.1, "What a Java EE Platform's Deployment Tools Must Do", a new stanza was added to place the policy context in service.
- Section 3.1.1.1, "Policy Context Life Cycle", was added.
- In Section 3.1.3, "Translating Servlet Deployment Descriptors", the call to getPolicyConfiguration was augmented with a second parameter to ensure that all policy statements are removed from the context.
- Section 3.1.3.1, "Translating security-constraint Elements", was rewritten such that the target names of the WebResourcePermission and WebUserDataPermission policy statements resulting from the translation are qualified such that they precisely specify the resources to which they apply. The most significant affect of this change is that it captures the best-matching semantics of the Servlet constraint model in the permission names, such that these permissions can be tested using the standard J2SE permission evaluation logic.
- Added a new section, "Qualified URL Pattern Names", to describe the rules for composing the target names used in the construction of the

WebResourcePermission and WebUserDataPermission policy statements resulting from the translation of Servlet security constraints.

- The section that had described the "Mapping to Unacceptable Transport Connection Types" was changed to describe the mapping to "acceptable" connection type. The title of the section was changed to "Mapping Transport Guarantee to Connection Type". Table 3-1 was also changed to reflect the change to "acceptable" connection types, and the connection type values in the table were modified to agree with the transportTypeSpec syntax of the WebUserDataPermission class.
- Section 3.1.3.3, "Servlet URL-Pattern Matching Rules", was added to support the pattern qualification section, and relevant sections of the enforcement subcontract.
- Section 3.1.3.4, "Example" was added
- In Section 3.1.5, "Translating EJB Deployment Descriptors", the call to getPolicyConfiguration was augmented with a second parameter to ensure that all policy statements are removed from the context.
- The last paragraph of Section 3.1.5.2, "Translating the EJB exclude-list", was clarified.
- Section 3.1.6, "Deploying an Application or Module", Section 3.1.7, "Undeploying an Application or Module", Section 3.1.8, "Deploying to an existing Policy Configuration", and Section 3.1.9, "Redeploying a Module", were all changed o reflect the introduction of the policy context life cycle and the commit method.
- The inService method was added to the factory methods called out in the first paragraph of Section 3.3, "Permission to Configure Policy", and the SecurityPermission required by these methods was changed from "getPolicy" to "setPolicy" to correct an inconsistency with the Java implementation.

C.7.6 Changes to Policy Decision and Enforcement Subcontract

- Section 4.1, "Policy Enforcement by Servlet Containers", was modified to require that containers use Policy to make access control decisions.
- Section 4.1.2, "Evaluation of Transport Guarantees", was modified to describe how the transport type value is obtained for the permission construction, and to reflect the change made to the WebUserDataPermission class such that it is no longer checked by "determining if a Permission has been excluded".

- Section 4.1.2, “Evaluation of Transport Guarantees”, and Section 4.1.3, “Pre-dispatch Decision”, were changed to reference the error processing defined in the Servlet specification.
- Section 4.2.1, “Servlet Policy Decision Semantics”, was rewritten to reflect the qualification of the permission names, and the change to conventional permission evaluation semantics.
- Section 4.2.1.3, “WebResourcePermission Matching Rules”, Section 4.2.1.4, “WebRoleRefPermission Matching Rules”, and Section 4.2.1.5, “WebUserDataPermission Matching Rules” were added to define the permission specific matching semantics necessary to support the policy decision semantics.
- Section 4.2.2.1, “Servlet URL-Pattern Matching Rules”, Section 4.2.2.2, “Servlet Constraint Matching Examples”, and Section 4.2.3, “WebRoleRefPermission Processing Semantics” were removed from the document, as the change to qualified pattern names made these sections unnecessary.
- Section 4.3, “Policy Enforcement by EJB Containers”, was modified to require that containers use Policy to make access control decisions.
- Section 4.4.1, “EJB Policy Decision Semantics”, was replaced with a simplified section that references Section 4.2.1, “Servlet Policy Decision Semantics”.
- Section 4.4.1.1, “EJBMethodPermission Matching Rules”, and Section 4.4.1.2, “EJBRoleRefPermission Matching Rules”, were added to define the permission specific matching semantics necessary to support the policy decision semantics. These new sections replaced Section 4.4.2, “EJB Permission Matching Rules”.
- The last paragraph of Section 4.5, “Component runAs Identity”, was modified to ensure that the AccessControlContext includes a SubjectDomainCombiner.
- In Section 4.6.1, “Policy Context Handlers”, changed the last sentence of the paragraph to “...if these actions will cause the container to fail in its processing of the associated request”.
- In Section 4.6.1.1, “Container Subject Policy Context Handler” replaced “caller’s identify” with “caller’s identity”.
- In Section 4.6.1.2, “SOAPMessage Policy Context Handler”, reduce to only EJB container, and added additional qualification of the request coming in at the ServiceEndpoint method interface.
- In Section 4.6.1.5, “EJB Arguments Policy Context Handler”, clarified that this handler may not be used if the request came in on the

ServiceEndpoint method interface. Also changed the return type when there are no arguments to an empty array.

- Renamed section Section 4.7, "Checking AccessControlContext Independent Grants" and changed it to reflect the changes made to WebUserDataPermissions such that they are no longer "excluded" permissions.
- In Section 4.9, "Missing Policy Contexts", changed replaced contains with inService method.

C.7.7 Changes to API

- A new class diagram was imported to reflect the changes to the API, most notably the introduction of the PolicyContextException class.
- The javadocs were regenerated to conceal implementation specific private instance variables.
- Added "ServiceEndpoint" to the list of alternative MethodInterface identifiers for EJBMethodPermissions.
- More completely specified EJBMethodPermission matching of methodNameSpec in implies
- Added policy context life cycle, including description, and state table to PolicyConfiguration interface.
- Added new methods "commit" and inService to the PolicyConfiguration interface.
- Changed all the method signatures of the PolicyConfiguration interface to throw PolicyContextException, and described the other exceptions that implementations are required to throw.
- Changed the documentation of getPolicyConfigurationFactory to properly identify the system property.
- Added a new parameter to the getPolicyConfiguration method of PolicyConfigurationFactory to indicate whether or not all the policy statements should be removed from the policy context.
- Renamed contains of PolicyConfigurationFactory class to inService.
- Changed all the method signatures of the PolicyConfigurationFactory class to throw PolicyContextException, and described the other exceptions that implementations are required to throw.
- Changed authorization requirement of the PolicyContext class to allow containers to be responsible for deciding how callers of this method must be authorized.
- Changed the getContext and registerHandler methods of the PolicyContext class to declare that they throw

PolicyContextException., and described the other exceptions that these methods are required to throw.
- Changed the format of the name used to construct a WebResourcePermission to contain a URLPatternSpec, and described the restrictions on the patterns appearing in the URLPatternList.
- Modified the specification of the implies and equals methods of WebResourcePermission to account for the URLPatternSpec.
- Changed the format of the name used to construct a WebUserDataPermission to contain a URLPatternSpec, and described the restrictions on the patterns appearing in the URLPatternList.
- Changed BNF for "actions" of WebUserDataPermission such that a separating ":" is not required if a transportType is not explicitly specified.
- Replaced transportTypeList in actions of WebuserdataPermission with a single transportType value.
- Modified the specification of the implies and equals methods of WebUserDataPermission to account for the URLPatternSpec.
- Comparable Interface was removed from WebResourcePermission and WebUserDataPermission.
- description of the second clause of the "servlet matching rules" of WebResourcePermission.implies and WebUserDataPermission.implies were changed to properly reflect the servlet matching semantics; where for example, /a/b/* must match /a/b in addition to /a/b/z.
- In WebUserDataPermission constructor removed extra "and" in "...by calling and HttpServletRequest.isSecure()".
- In description of PolicyContextHandler.getContext, removed extra "the" from "and obtain from it the the".

C.7.8 Changes to References

- Upgraded document version references for [J2EE specification], [J2SE specification], [EJB specification], and [Servlet specification] to 1.4, 1.4.0, 2.1, and 2.4 respectively. Also updated URL for [J2EE specification].

C.7.9 Changes to Issues

- Added new issue, B.13,"Implementing Web Security Constraints as Permission".
- Added new issue, B.14,"Exception Handling".

- Added new issue, B.15,“PolicyConfiguration Commit".
- Added new issue, B.16,“Support for ServiceEndpoint methodInterface".

C.8 Changes in Proposed Final Draft 2 Expert Draft 2

C.8.1 Changes to Preface

- fixed typos, and added additional RI team member to credits.

C.8.2 Changes to Policy Configuration Subcontract

- In Section 3.1.3.3, “Servlet URL-Pattern Matching Rules”, added additional clause to support universal matching by “/*”.
- In Section 3.1.3.4, “Example”, Added comments to security-constraint elements, Also corrected qualified URL Pattern Names occurring in TABLE 3-3 and TABLE 3-4.
- In Section 3.1.6, “Deploying an Application or Module”, changed the text of the footnote to properly reflect that policy contexts are linked by object not by identifier.

C.8.3 Changes to Policy Decision and Enforcement Subcontract

- In Section 4.1.2, “Evaluation of Transport Guarantees”, and Section 4.1.3, “Pre-dispatch Decision”, changed the corresponding construction descriptions to be less prescriptive such that calling any constructor that results in the proper name being established would be allowed. Also indicated that the resulting url-pattern is to be “unqualified”.
- Modified Section 4.2.1, “Servlet Policy Decision Semantics”, to require that the policy statements of the default policy context be included in the access decisions and to require that the subject based policy statements be tested when the status is unresolved following the excluded and unchecked evaluations.
- Added a new Section 4.2.1.1, “Matching Qualified URL Pattern Names” to describe URLPatternSpec matching, and replaced the duplicate descriptions of this processing in sections Section 4.2.1.3, “WebResourcePermission Matching Rules” and Section 4.2.1.5, “WebUserDataPermission Matching Rules” with a reference to this new section. Also modified the description of the comparison to support symmetric implication as necessary to support consistent

semantics between the implies and equals methods of these permissions.

- Added requirement that the comparisons defined by Section 4.2.1.3, "WebResourcePermission Matching Rules", Section 4.2.1.4, "WebRoleRefPermission Matching Rules", Section 4.2.1.5, "WebUserDataPermission Matching Rules", Section 4.4.1.1, "EJBMethodPermission Matching Rules", and Section 4.4.1.2, "EJBRoleRefPermission Matching Rules" be case sensitive.
- The word "form" was changed to "from" in first paragraph of Section 4.7, "Checking AccessControlContext Independent Grants".
- In bullets 4 and 5 of Section 4.7, "Checking AccessControlContext Independent Grants", removed "that was constructed without static permissions and".
- Rewrote Section 4.10, "Default Policy Context" to indicate describe the properties of the default policy context, and to require that its policy statements be included in every access decision.

C.8.4 Changes to API

- comments on HttpServletRequest based constructors for WebResourcePermission and WebUserDataPermission were changed so as not to imply that this is the only constructor that may be used by a container "prior to checking" a Servlet request.
- the description of the implies method of WebResourcePermission and WebUserDataPermission was modified to support the maxim that two permission objects p1 and p2 are equivalent iff p1.implies(p2) and p2.implies(p1). To do so required handling the case where the name of the argument permission (to implies) is a qualified URLPatternSpec.
- the description of the servlet matching rules in the implies method of WebResourcePermission and WebUserDataPermission was corrected to account for universal matching by "/*".

C.9 Changes in Proposed Final Draft 2 Expert Draft 3

C.9.1 Changes to Policy Configuration Subcontract

- Added a new first paragraph to Section 3.1.3.1, "Translating security-constraint Elements", to describe the treatment of patterns overridden

by and made irrelevant by the presence of the "/*" pattern in the a web-resource-collection within the deployment descriptor.

- Moved the last paragraph in "Qualified URL Pattern Names" to be its first, and added a new paragraph to its end to describe irrelevant patterns and their treatment by the permission constructors. Clarified the syntax and description of URLPattern qualification. Indicated that patterns qualified by other qualifying patterns may be dropped from the list of qualifying patterns (and described why).
- In Section 3.1.3.4, "Example", removed the "/*" pattern from the first web-resource-collection of the first security constraint, and made the corresponding changes to the table of qualified URL pattern names and the table of constructed permissions.
- Added a new column to TABLE 3-3 of Section 3.1.3.4, "Example" to represent the canonical form of the qualified names.The description of TABLE 3-4 was modified to indicate that the names in its second column were obtained from the first column of TABLE 3-3, and that any equivalent form of the qualified names, including their canonical forms, could have been used in the permission constructions.

3.9.2 Changes to Policy Decision and Enforcement Subcontract

- In Section 4.1.2, "Evaluation of Transport Guarantees", clarified the actions value used for a request that arrives on an unprotected connection.

C.9.3 Changes to API

- The URLPatternList descriptions of the WebResourcePermission and WebUserDataPermission classes; were modified to require that no pattern in a URLPatternList may imply the first pattern of the URLPatternSpec, as otherwise the URLPatternSpec could not imply itself which would violate the required equals semantics.
- The definition of the equals method of the WebResourcePermission and WebUserDataPermission classes; was modified such that different URLPatternList values are equal if the lists imply the same patterns.

C.10 Changes in Proposed Final Draft 2 Expert Draft 4

C.10.1 Changes to API

- The serialization (see Serialized Form on html Javadocs) of the javax.security.jacc permission classes was described more completely and to remove unnecessary constraints on implementations.
- The canonical forms produced by the getActions methods of the WebResourcePermission and WebUserDataPermission classes were more completely specified.

C.11 Changes in Final Release

C.11.1 Changes to License

- License was replaced

C.11.2 Changes to the Preface

- The preface was changed to reflect the purpose of the Final Release.
- Additional contributor names were added.

C.11.3 Changes to Overview

- Added requirement to support Section 4.8, "Checking the Caller for a Permission", to ensure that policy providers not place extra requirements on containers.

C.11.4 Changes to Provider Configuration Subcontract

- Added another catch clause to the code sample in Section 2.7, "What the Application Server Must Do", to support verification that the loaded object is an instanceof javax.security.Policy.

C.11.5 Changes to Policy Configuration Subcontract

- Added definition of what it means for two translations to be "equivalent" to Section 3.1, "What a Java EE Platform's Deployment Tools Must Do".

- Added clarification to Section 3.1.3.1, “Translating security-constraint Elements” to allow for “equivalent” translations.
- Restated the translation description of Section 3.1.3.1, “Translating security-constraint Elements”, such that it no longer prescribes the number of permissions that must be constructed.
- Modified the title of the second column of Table 3-1.
- Restated the translation description of Section 3.1.3.2, “Translating Servlet security-role-ref Elements”, such that it no longer is as prescriptive with respect to the “construction” of permissions, and such that it defines the name to use for the “additional” permissions.
- Fixed a syntax problem, missing "<" in "urlPattern>", in Section 3.1.3.4, “Example”.
- Changed some of the actions values of Table 3-4, such that they are all in canonical form. Added table footnote to that effect.
- Added clarification to Section 3.1.5.1, “Translating EJB method-permission Elements” to allow for “equivalent” translations.
- Restated the translation description of Section 3.1.5.1, “Translating EJB method-permission Elements”, such that it no longer such that it no longer prescribes the number of permissions that must be constructed.
- Clarified the linking requirements of Section 3.1.6, “Deploying an Application or Module” and of Section 3.1.9, “Redeploying a Module”.
- In Section 3.1.7, “Undeploying an Application or Module”, Section 3.1.8, “Deploying to an existing Policy Configuration”, and in Section 3.1.9, “Redeploying a Module”, changed “must stop accepting” to “must stop dispatching” requests.

C.11.6 Changes to Policy Decision and Enforcement Contract

- Added special rule for checking "/" to Section 4.1.2, “Evaluation of Transport Guarantees”, and Section 4.1.3, “Pre-dispatch Decision”.
- In Section 4.1.2, “Evaluation of Transport Guarantees”, Section 4.1.3, “Pre-dispatch Decision”, Section 4.1.4, “Application Embedded Privilege Test”, Section 4.3.1, “EJB Pre-dispatch Decision”, and Section 4.3.2, “EJB Application Embedded Privilege Test”, changed the description of how the checked permission is "obtained".
- Added clarification of "the scope of a containers processing of a component request" to Section 4.6.1, “Policy Context Handlers”.
- Added a clarification to Section 4.6.1, “Policy Context Handlers”, allowing containers to delay the registration of the required handlers.

- In Section 4.6.1.4, “EnterpriseBean Policy Context Handler”, restricted the use of this handler to the business method of the EJB Remote, Local, or ServiceEndpoint interfaces of the EnterpriseBean object.
- Added a footnote to Section 4.8, “Checking the Caller for a Permission”, to clarify why calling Policy.getPermissions is not recommended.
- Added Section 4.12, “Optimization of Permission Evaluations” to describe the circumstances under which containers may caching the results of permission evaluations.

C.11.7 Changes to API

- Added package description
- Changed MethodSpec and constructor descriptions of EJBMethodPermission to provide support for additional method-intf values.
- Clarified the syntax of typeName as used in methodParams of EJBMethodPermission. Also specified the corresponding affect on the canonical form returned by getActions.
- For both WebResourcePermission nd WebUserDataPermission, specified the effect of constructing these permissions with a null name. Also clarified that the empty string is a supported exact pattern.
- For both WebResourcePermission nd WebUserDataPermission, corrected definition of HttpServletRequest based constructors such that they obtain the permission name from the RequestURI minus the contextPath, except for the special case where the name would be "/", in which case the empty string is used as the permission name.
- In WebUserDataPermission, Fixed errors in the BNF for transportType.
- Added text to javadoc of JACC permission classes to make it clear that these permissions may implement newPermissionCollection or inherit its implementation from their superclass.
- Modified the definition of the PolicyContext class to allow for implementations that restrict access to the security sensitive methods of this utility class without necessarily resorting to checking the setPolicy SecurityPermission.

C.11.8 Changes to Appendix A: Related Documents

- Updated the copyright dates.

C.11.9 Changes to Appendix B: Issues

- Added descriptions of 3 new issues: Section B.17, “TypeNames of EJBMethodPermission Array Parameters”, Section B.18, “Checking Permission on the root of a Web Application”, and Section B.19, “Calling isUserInRole from JSP not mapped to a Servlet”.

C.12 Changes in Errata A

C.12.1 Changes to Policy Configuration Subcontract

- Page 24: added requirement to “Translating Servlet security-role-ref Elements" for extra WebRoleRefPermission objects to be created to support calls to isUserInRole from unmapped JSPs.

C.12.2 Changes to Policy Enforcement Subcontract

- Page 37: added requirement to “Application Embedded Privilege Test" to support calling isUserInRole from an unmapped (to servlet) web resource.
- page 47: added footnote to “Checking the Caller for a Permission" to act as a forward reference to optimization by reuse of unauthenticated results as allowed for by new text added to “Optimization of Permission Evaluations". This optimization allows a container to optimize authorization checks on unprotected resources.
- Page 50: added new clarifying text to “Optimization of Permission Evaluations" to support performance optimization based on reuse of evaluation results. In addition to reuse of equivalent evaluations, added text to support reuse of unauthenticated evaluations to authorize evaluations independent of caller identity. Described a common practice that could be implemented by containers and providers, and that would cause containers to be notified by providers of policy changes. By following the suggested practice providers would be able to tell when containers expect to be notified, for containers to determine if they will be notified, and for containers to determine if their provider has other properties necessary to sustain reuse.

C.12.3 Changes to API

- Page 87: Clarified Description of WebRoleRefPermission class.

- Page 88: Modifed description of name parameter of WebRoleRefPermission constructor to describe use of empty-string name.

C.12.4 Changes to Appendix B: Issues

- Page 105: removed sentence from description of resolution of issue B19, “Calling isUserInRole from JSP not mapped to a Servlet", that had indicated that the resolution would NOT be adopted until the Servlet spec was changed. As a result of this errata, the resolution to issue B19 has been fully integrated.

C.13 Changes in Errata B

C.13.1 Changes to Overview

- Page 7: modified requirement 9 to allow for and describe the circumstances under which a container may run without a SecurityManager.
- Page 8: added Section 1.5, “Running Without a SecurityManager” to describe the changes to this contract that apply to containers running without a J2SE SecurityManager.

C.14 Change log for Errata C

C.14.1 Changes Made Throughout the Document

- Changed the "J2EE" and "J2SE" platform names (when not used with a specific version such as J2EE 1.4) to "Java EE" and "Java SE" respectively.
- Changed improper uses of "affect" to "effect".

C.14.2 Changes to Overview

- In “Assumptions” on page 5", clarified assumptions 1 and 3 to indicate that contract is intended to apply and be required by future versions of the Java EE platform.

C.14.3 Changes to Provider Configuration Contract

- Generalized the J2EE 1.4 version specific requirements such that they also apply to later versions of the EE platform.

C.14.4 Changes to Policy Configuration Contract

- Extended the chapter abstract to indicate that the subcontract applies to the configuration of policy providers from authorization rules defined within Java code using common annotations.
- In "What a Java EE Platform's Deployment Tools Must Do" on page 17 and 18, described the deployment tool requirements relating to annotation processing, and the merging of annotations into the deployment descriptor such that the translation may occur using the deployment descriptor translation rules.
- In "Servlet Policy Context Identifiers" on page 21, described why each module of a multi-module web application must be deployed to a separate policy context.
- In "Translating Servlet security-role-ref Elements" on page 26, clarified that the set of all roles defined for the application is used to determine the additional permissions to be constructed.
- In "EJB Policy Context Identifiers" on page 31, added rule to ensure that no two EJBs in a policy context share the same ejb-name. If this rule is not observed the policy statements for the EJBs would be inappropriately combined.

C.14.5 Changes to Policy Decision and Enforcement Contract

- Inserted new section "Permission Names for Transport and Pre-Dispatch Decisions" on page 37, to call attention to the description of how the corresponding permissions names are constructed. This section was intended to account for the welcome file processing defined by the Servlet specification. The corresponding clarification of the relationship between welcome file processing and servlet-constraint processing was not made to the Servlet spec, so, consistent with the assumptions under which this spec. was defined, clarifying semantics will not be prescribed by this spec. until they are adopted by the Servlet specification.

- Revised section “Evaluation of Transport Guarantees” on page 37 and section “Pre-dispatch Decision” on page 38, to refer to the newly inserted section for the definition of their respective permission names.
- Added new sentence the description of the “EnterpriseBean Policy Context Handler” on page 48 to account for EJB 3.0 Session and Entity beans which are not required to implement the javax.ejb.EnterpriseBean interface.

C.14.6 Changes to API

- On page 69, clarified the description of the PolicyConfiguration.commit() method to indicate that it also throws an UnsupportedOperationException when completing the commit would cause there to be two or more inService and linked policy contexts with different principal-to-role mappings.
- Changes to the description of the HttpServletRequest based constructors of the WebResourcePermission and WebUserDataPermission intended to clarify that welcome file processing must have been performed before permission construction were deferred pending clarification of the corresponding functionality in the Servlet Specification

C.15 Change log for Errata D

C.15.1 Changes Made Throughout the Document

- Changed The specification version from 1.0 to 1.1

C.15.2 Changes to Policy Configuration Contract

- Ammended Section 3.1.3.1, “Translating security-constraint Elements” to support the translation of security-constraints containing extension methods as defined in IETF RFC 2616 "Hypertext Transfer Protocol -- HTTP/1.1".
- Added a new subsection, “HTTP Method Exception List", to describe the representation of non-enumerable HTTP method subsets as necessary, for example, to identify all methods not named in a security-constraint.
- Modified the actions entries in Table 3-4: "Permissions and PolicyConfiguration Operations from Example" to conform to the

translation changes required to support non-enumerable http extension methods.

C.15.3 Changes to Policy Decision and Enforcement Contract

- Inserted new Section 4.2.1.2, “Matching HTTP Method Specifications” to describe the HTTPMethodSpec as revised (by the definition of the HTTPMethodExceptionList) to support HTTP extension methods.
- Modified Section 4.2.1.3, “WebResourcePermission Matching Rules” and Section 4.2.1.5, “WebUserDataPermission Matching Rules” to refer to the new section describing the matching of HTTP method specifications.

C.15.4 Changes to API

- Modified the WebResourcePermission class to support HTTP extension methods. Extended the permission's actions syntax to represent HTTP method exception lists so that non-enumerable method subsets can be represented in the permission's actions. Exception lists are used to represent unconstrained http method subsets.
- Modified the WebUserDataPermission class to support HTTP extension methods. Extended the permission's actions syntax to represent HTTP method exception lists as was done for the WebResourcePermission class.

C.15.5 Changes to Appendix B: Issues

- Added new issue Section B.20, “Support for HTTP Extension Methods”. Resolution describes consequences with respect to backward compatibility:
- Added new issue Section B.21, “Welcome File and security-constraint Processing” to describe the need for clarification of the relationship between welcome file processing, which can change the effective request URI, and the url-patterns applied in security-constraint processing.
- Added new issue Section B.22, “Colons Within path-segment of Request URI” to document the potential ambiguity resulting from the use, by the WebResourcePermission and WebUserDataPermission classes, of the colon character to distinguish qualifiying patterns.

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
650 960-1300

For U.S. Sales Office locations, call:
800 821-4643
In California:
800 821-4642

Australia: (02) 844 5000
Belgium: 32 2 716 7911
Canada: 416 477-6745
Finland: +358-0-525561
France: (1) 30 67 50 00
Germany: (0) 89-46 00 8-0
Hong Kong: 852 802 4188
Italy: 039 60551
Japan: (03) 5717-5000
Korea: 822-563-8700
Latin America: 650 688-9464
The Netherlands: 033 501234
New Zealand: (04) 499 2344
Nordic Countries: +46 (0) 8 623 90 00
PRC: 861-849 2828
Singapore: 224 3388
Spain: (91) 5551648
Switzerland: (1) 825 71 11
Taiwan: 2-514-0567
UK: 0276 20444

Elsewhere in the world,
call Corporate Headquarters:
650 960-1300
Intercontinental Sales: 650 688-9000