JAKARTA EE

Jakarta Authorization

Jakarta Authorization Team,
https://projects.eclipse.org/projects/ee4j.authorization

3.0, April 07, 2024: Final

Table of Contents

```
Specification: Jakarta Authorization

Version: 3.0

Status: Final

Release: April 07, 2024
```

Eclipse Foundation Specification License - v1.1

By using and/or copying this document, or the Eclipse Foundation document from which this statement is linked or incorporated by reference, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to copy, and distribute the contents of this document, or the Eclipse Foundation document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

- link or URL to the original Eclipse Foundation document.
- All existing copyright notices, or if one does not exist, a notice (hypertext is preferred, but a textual representation is permitted) of the form: "Copyright (c) [$date-of-document] Eclipse Foundation AISBL <<url to this license>> "

Inclusion of the full text of this NOTICE must be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of Eclipse Foundation documents is granted pursuant to this license, except anyone may prepare and distribute derivative works and portions of this document in software that implements the specification, in supporting materials accompanying such software, and in documentation of such software, PROVIDED that all such works include the notice below. HOWEVER, the publication of derivative works of this document for use as a technical specification is expressly prohibited.

The notice is:

"Copyright (c) [$date-of-document] Eclipse Foundation AISBL. This software or document includes material copied from or derived from [title and URI of the Eclipse Foundation specification document]."

Disclaimers

.1. Preface

.2. Status of Document

This document is the Final Release of the Jakarta Authorization 3.0 specification and represents the definition of this technology as implemented by a compatible implementation (CI) and verified by the technology compatibility kit (TCK) . This specification was developed under the EFSP (https://www.eclipse.org/projects/efsp/).

.3. Audience

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

.4. Abstract

This specification defines new java.security.Permission classes to satisfy the Jakarta 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.

.5. 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].

.6. 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

Paul Hendley Sun Microsystems

Kumar Jayanti Sun Microsystems

Eric Jendrock Sun Microsystems

Jagadesh Babu Munta Sun Microsystems

Tony Ng Sun Microsystems

Craig Perez Oracle America, Inc.

Raja Perumal Sun Microsystems

Deepa Singh Sun Microsystems

Harpreet Singh Sun Microsystems

Nithya Subramanian 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

After transfer to the Eclipse Foundation, the following people have enhanced the specification further:

Arjan Tijms

Guillermo González de Agüero

Jean-Louis Monteiro

Darran Lofthouse

Chapter 1. Overview

This specification defines a contract between Jakarta 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, Policy Configuration Subcontract, and the Policy Decision and Enforcement Subcontract.

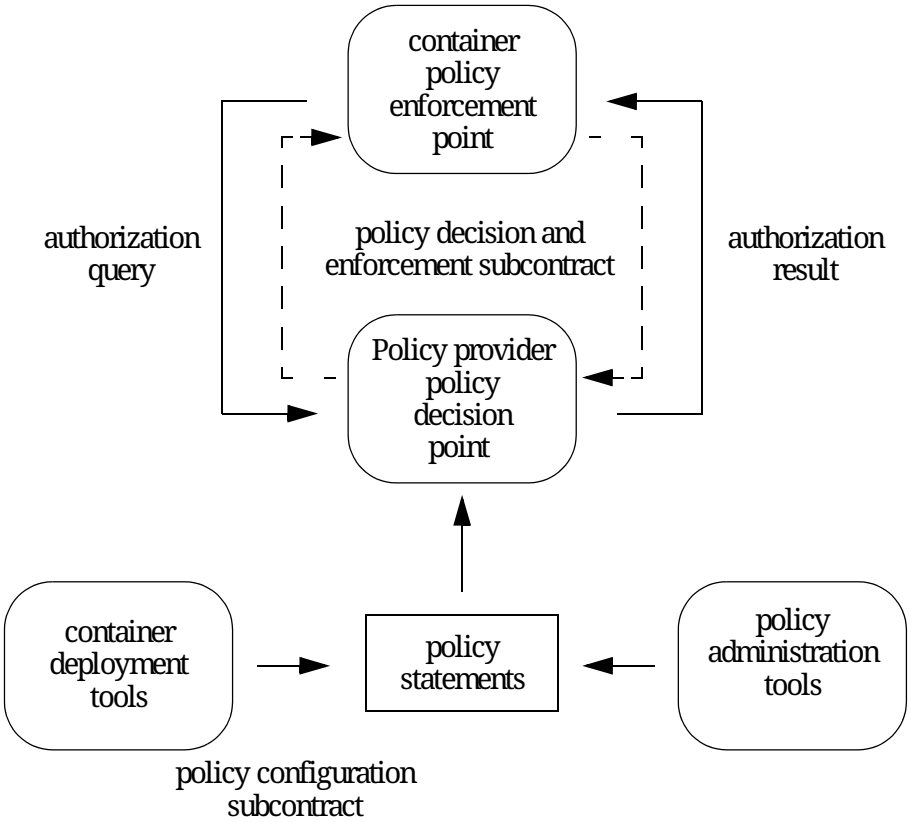

Figure 1-1 Policy Configuration and Enforcement Subcontracts

1.2. Relationship to other specifications

1.2.1. Relationship to Jakarta Security

Jakarta Security defines an end-developer focused API for securing applications in Jakarta EE using modern paradigms. It takes advantage of Jakarta CDI to provide an ease of use development experience. Jakarta Security uses this specification for its lower level authorization functionality. As such, this specification provides the SPI (Service Provider Interface) between a Jakarta EE Environment and an implementation of this specification, while Jakarta Security provides high

level ease of use functionality on top of that.

1.2.2. Relationship to Jakarta Authentication

Jakarta Authentication defines a lower-level SPI for plugable authentication modules into a Jakarta EE environment. It's similar to this specification in providing an SPI, with Jakarta Security providing high level ease of use functionality on top of that, with the obvious difference that Jakarta Authentication focuses on the authentication concerns of security, and this specification does so for the authorization concerns.

1.2.3. Relationship to Jakarta Servlet

Jakarta Servlet provides classes that respond to HTTP requests including a security model and the concept of a (web) application and deployable archives (wars).

This specification does exactly what a Jakarta Servlet implementation does as well; storing a representation of its security declarations (i.e. the constraints from its web.xml) internally, and using these to enforce the pre-dispatch security (access to a resource), and to support programmatic security (such as `HttpServletRequest.isUserInRole`).

Whereas in Jakarta Servlet this process is an internal implementation detail, this specification standardizes it. As a result, a Jakarta Servlet container not using Jakarta Authorization, and a Jakarta Servlet container delegating to Jakarta Authorization should have the exact same security behaviour when the defaults of Jakarta Authorization are used.

Jakarta Authorization expands on the Servlet security model, by making the objects involved in the authorization enforcement replaceable.

1.2.4. Relationship to Jakarta Enterprise Beans

Jakarta Enterprise Beans is an alternative component model in Jakarta EE. As of Jakarta EE 11 this model is strongly de-emphasised and may eventually be phased out entirely.

Like Jakarta Servlet, Jakarta Enterprise Beans defines its own security model which this specification supports by standardizing how its constraints are stored and enforced.

1.3. Terminology

Jakarta EE application

: A collection of one ore more Jakarta 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.

Application Server

For the purpose of this specification; a Jakarta EE environment capable of running multiple Jakarta EE applications at the same time where each Jakarta EE applications can consist of one or more modules (e.g. an EAR containing .war and .jar modules), and also supporting the deploy, undeploy and redeploy processes for each such application. For example, Eclipse GlassFish or Red Hat JBoss EAP.

Jakarta Authorization implementation

The code that most directly implements this specification, and which can be integrated into an Application Server or other Jakarta EE environment or Jakarta EE-like environment (like a standalone Servlet Container, such as Tomcat). A Jakarta Authorization implementation can be a standalone (modular) .jar file, potentially from a separate project, or it can be highly integrated into the code of a specific Jakarta EE environment.

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.

Policy interface

The `jakarta.security.jacc.Policy` abstract class.

Policy provider

An instance of a class that implements the 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.

1.4. Assumptions

1. The contract defined in this specification must be applicable to Jakarta EE 11 and future versions of the Jakarta EE platform.
2. We are defining a contract to be satisfied by Jakarta Authorization's own Policy providers.
3. Jakarta EE 11 platforms will be required to implement the contract defined by this specification. This contract will be a required element of subsequent versions of the Jakarta EE platform.
4. Jakarta EE application roles will be modelled as collections of permissions that are granted to principals.
5. 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()`).
6. This contract will shift the responsibility for performing all of the authorization decisions pertaining to a Jakarta 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
7. It is assumed that there are providers that are unable to enumerate all the permissions that pertain to a subject before returning from `Policy.getPermissionCollection()`.
8. 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.
9. Where the Jakarta Servlet specifications is incomplete or ambiguous in its 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 committers of the corresponding component specification.

1.5. Requirements

1. This contract must support providers that are unable to determine, before returning from `Policy.getPermissionCollection()`, all the permissions that pertain to a subject.
2. Each Policy provider that satisfies this contract must perform or delegate to another provider just the permission evaluations requested via its interface to implement Jakarta 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 Jakarta Enterprise Beans and Jakarta Servlet specifications corresponding to the target platform. The provider is not required to satisfy the authorization requirements pertaining to any of the above specifications for which the target platform is not a compatible implementation.
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 Jakarta 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 Jakarta 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 Jakarta Servlet specification.
8. To support this contract in a Jakarta Servlet environment, a container or its deployment tools must create policy statements as necessary to support Servlet's "default role-ref semantic".
9. Policy providers must perform the permission evaluations corresponding to container pre-dispatch decisions and application embedded privilege tests (i.e `isUserInRole` and `isCallerInRole`).

1.6. Non Requirements

1. This specification does not require that containers support server-side authentication module plug-ins (for example, those from Jakarta Authentication) for the purpose of populating subjects with authorization provider specific principals.
2. This specification does not require that subjects be attributed with role principals as a result of authentication.
3. This specification 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 specification does not require that providers support a policy syntax for granting to principals roles as collections of permissions.
5. Although the specification is focused on defining permissions and policy for use by Jakarta 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 specification to extend or modify the Jakarta EE authorization model to be equivalent to standard RBAC models for access control.

1.7. Jakarta Servlet or Jakarta Enterprise Beans only containers

The requirements of this specification that must be satisfied by a target platform that is a

compatible implementation of one but not both of the Jakarta Servlet and Jakarta Enterprise Beans specifications are reduced as described in the next two sections.

1.8. Jakarta Servlet Only Containers

A platform that is a compatible implementation of the Jakarta Servlet specification and that is not a compatible implementation of the Jakarta Enterprise Beans specification must satisfy all of the requirements of this specification with the following exceptions:

1. the policy configuration requirements defined in Jakarta Enterprise Beans Policy Context Identifiers and in Translating Jakarta Enterprise Beans Deployment Descriptors
2. the policy enforcement requirements defined in Policy Decision and Enforcement Subcontract for Jakarta Enterprise Beans and Provider Support for Jakarta Enterprise Beans Policy Enforcement
3. the policy context handler requirements defined in SOAPMessage Policy Context Handler, and EnterpriseBean Policy Context Handler, and Jakarta Enterprise Beans Arguments Policy Context Handler

1.9. Jakarta Enterprise Beans Only Containers

A platform that is is a compatible implementation of the Jakarta Enterprise beans specification and that is not a compatible implementation of the Jakarta Servlet specification must satisfy all of the requirements of this specification with the following exceptions:

1. the policy configuration requirements defined in Servlet Policy Context Identifiers and in Translating Servlet Deployment Descriptors
2. the policy enforcement requirements defined in Policy Enforcement by Servlet Containers and Provider Support for Servlet Policy Enforcement
3. the policy context handler requirements defined in HttpServletRequest Policy Context Handler

Chapter 2. Provider Configuration Subcontract

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

2.1. Policy Implementation Class

The contract defined by this specification has been designed to work in Jakarta EE 11 or later Jakarta EE environments with the default `jakarta.security.jacc.Policy` implementation class.

An application server that supports this contract must allow replacement of the top level `jakarta.security.jacc.Policy` object used by the application server for all applications running on it, in addition to supporting a `jakarta.security.jacc.Policy` object for individual applications.

Replacement is done via the `jakarta.security.jacc.PolicyFactory` abstract factory class. A default static method, `getPolicyFactory` is provided that uses the system property `jakarta.security.jacc.PolicyFactory.provider` to locate a concrete implementation. The container or an application can alternatively set a custom `PolicyFactory` using the `setPolicyFactory` method. In that case the `PolicyFactory` implementation can come from a container specific configuration, or in case of the Servlet Container from the web application's servlet context initialization parameter (context-param in web.xml) `jakarta.security.jacc.PolicyFactory.provider`.

If the `PolicyFactory` has a public constructor with one argument of type `PolicyFactory`, then the container should call this constructor with as argument the `PolicyFactory` instance that is being replaced. This allows a replacement `PolicyFactory` to wrap the existing one and selectively provide extra functionality.

From this factory class a concrete implementation of the Policy of type `jakarta.security.jacc.Policy` can be obtained using the method `getPolicy`.

In addition to replacing the `PolicyFactory`, the default `PolicyFactory` must also allow replacing just the `Policy` instance. This is detailed below in What the Jakarta Authorization Implementation Must Do.

2.2. Permission Implementation Classes

This contract defines the package, `jakarta.security.jacc`, that contains (among other things) `Permission` classes to be used by containers in their access decisions.

2.3. Policy Configuration Interface

A Policy Configuration implementation is the object that holds the Permission instances a Policy can use to make its authorization decisions. The Policy Configuration implementation can be replaced if needed, although for most common uses this should not be needed.

Replacement is done via the `jakarta.security.jacc.PolicyConfigurationFactory` abstract factory

class. A default static method, `getPolicyConfigurationFactory` is provided that uses the system property `jakarta.security.jacc.PolicyConfigurationFactory.provider` to locate a concrete implementation. The container can alternatively set a custom `PolicyConfigurationFactory` using the `setPolicyConfigurationFactory` method. In that case the `PolicyConfigurationFactory` implementation can come from a container specific configuration, or in case of the Servlet Container from the web application's servlet context initialization parameter (context-param in web.xml) `jakarta.security.jacc.PolicyConfigurationFactory.provider`.

If the `PolicyConfigurationFactory` has a public constructor with one argument of type `PolicyConfigurationFactory`, then the container should call this constructor with as argument the `PolicyConfigurationFactory` instance that is being replaced. This allows a replacement `PolicyConfigurationFactory` to wrap the existing one and selectively provide extra functionality.

From this factory class a concrete implementation of the Policy Configuration of type `jakarta.security.jacc.PolicyConfiguration` can be obtained using the method `getPolicyConfiguration`.

Use of the `PolicyConfiguration` interface is defined in Policy Configuration Subcontract.

2.4. PolicyContext Class and Context Handlers

This `jakarta.security.jacc` package defines a utility class that is used by containers to communicate policy context identifiers to Policy providers. The utility class is `jakarta.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 Policy Contexts and Policy Context Identifiers.

In addition to the methods used to communicate policy context identifiers, the `jakarta.security.jacc.PolicyContext` class also provides static methods that allow container specific context handlers that implement the `jakarta.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.

Use of the `PolicyContext` class is defined in Policy Configuration Subcontract.

2.5. PrincipalMapper Interface

A `jakarta.security.jacc.PrincipalMapper` is an object that maps from a collection of generic Principals or a `Subject` to well known entities in Jakarta EE. A `Subject` is a "bag of principles", which can contain any number of principals, with each principal in that bag representing some attribute

of the subject in question. A portable `Policy` does not know which principal represents what entity, as historically no interfaces or annotations have been standardised to mark them as such.

Using the `PrincipalMapper` a given number of well known entities can be retrieved from the `Subject`.

The following well known entities are supported:

- The caller principal - a `java.security.Principal` containing the name of the current authenticated user (if any).
- The role - a `java.lang.String` representing the logical application role associated with the caller principal.

A `PrincipalMapper` is intended to be used by a `Policy`, but should work outside a `Policy` (for instance, during request processing in a Servlet container).

2.6. What a Provider Must Do

A Jakarta Authorization implementation can be provided with custom classes that implement the `PolicyConfigurationFactory` class and `PolicyConfiguration` interface. These classes can be used by the `Policy` implementation class installed for use by the Jakarta Authorization implementation. In the case where the provider is not seeking to replace the `Policy` implementation used by the Jakarta Authorization implementation, no other components need be provided.

If the provider is seeking to replace the `Policy` implementation used by the Jakarta Authorization implementation, then the Jakarta Authorization implementation must be provided with an environment specific `Policy` implementation class.

A replacement `Policy` object must assume responsibility for performing all policy decisions within the entire application server (in case the Jakarta Authorization implementation is integrated into an application server) in which it is installed that are requested by way of the `Policy` interface that it implements. Alternatively or additionally a replacement `Policy` can be installed for an individual (web) application.

A replacement `Policy` object may accomplish this by delegating `jakarta.security.jacc` policy decisions to the corresponding default `Policy` implementation class. A replacement `Policy` object that fully or partially relies in this way on the corresponding default `Policy` implementation class must identify itself in its installation instructions as a “delegating Policy provider”.

2.7. What the Jakarta Authorization Implementation Must Do

A Jakarta Authorization implementation must bundle or install the Jakarta Authorization API. This API consists of all types from the `jakarta.security.jacc` package. It must provide default implementations of its interfaces and abstract classes.

To enable delegation of `jakarta.security.jacc` policy decisions to default `Policy` providers, all Jakarta Authorization implementations must support the following `Policy` replacement algorithm. The intent of the algorithm is to ensure that `Policy` objects can capture the instance of the

corresponding default `Policy` object during their integration into a container and such that they may delegate policy evaluations to it.

For each Jakarta EE 11 or later version Jakarta EE compatible implementation, if the system property "`jakarta.security.jacc.policy.provider`" is defined, the Jakarta Authorization implementation must construct an instance of the class identified by the system property, confirm that the resulting object is an instance of `jakarta.security.jacc.Policy`, and set, by calling the `jakarta.security.jacc.PolicyFactory#setPolicy` method, the resulting object as the corresponding `Policy` object used by the Jakarta Authorization implementation.

For example:

```
String policyClassName = System.getProperty("jakarta.security.jacc.policy.provider");

if (policyClassName != null) {
    try {
         Policy policy = loadPolicy(policyClassName);
         PolicyFactory.getPolicyFactory().setPolicy(policy);
     } catch (Exception e) {
         // ...
     }
}

private Policy loadPolicy(String policyClassName) throws ReflectiveOperationException,
SecurityException {
        Object policyInstance =
                Thread.currentThread()
                      .getContextClassLoader()
                      .loadClass(policyClassName)
                      .getDeclaredConstructor()
                      .newInstance();

        if (!(policyInstance instanceof Policy)) {
            throw new RuntimeException("..."));
        }

        return (Policy) policyInstance;
    }
```

Even when a Jakarta Authorization implementation has used the system property defined in this section to replace a `Policy` object used by the Jakarta Authorization implementation, the Jakarta Authorization implementation MUST be prepared for an individual web application to replace the `Policy` object once again.

For example:

```
@WebListener
public class PolicyRegistrationListener implements ServletContextListener {
```

```
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        PolicyFactory policyFactory = PolicyFactory.getPolicyFactory();
        policyFactory.setPolicy(new TestPolicy(policyFactory.getPolicy()));
    }
}
```

The Jakarta Authorization implementation MAY forbid setting the `Policy` by an application after that application has been taken into service (starting to process requests).

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

Chapter 3. Policy Configuration Subcontract

The Policy Configuration Subcontract defines the interactions between container deployment tools and providers to support the translation of declarative Jakarta EE authorization policy (i.e. constraints in web.xml) into policy statements (Permission instances) within a Jakarta Authorization Policy provider.

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

3.1. What a Jakarta EE Platform's Deployment Tools Must Do

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

```
PolicyConfigurationFactory policyConfigurationFactory =
    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 petPolicyConfiguration =
    policyConfigurationFactory.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 `jakarta.security.jacc` Permission classes.

```
WebResourcePermission webResourcePermission =
    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.

```
petPolicyConfiguration.addToRole("customer", webResourcePermission);
```

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

```
petPolicyConfiguration.linkConfiguration(petFoodPolicyConfiguration);
```

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.

```
petPolicyConfiguration.commit();
```

Independent of this specification, Jakarta EE deployment tools must translate and complete the declarative policy statements appearing in deployment descriptors into a form suitable for securing applications on the platform. These 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 Jakarta Servlet and Jakarta EE platform specifications.

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 Jakarta Authorization forms defined by this subcontract.

Two translations are equivalent if they produce corresponding collections of unchecked, excluded, and role permissions, and if all of the permissions of each such collection are implied[1] by the permissions of the corresponding or excluded collection of the other translation.

Translation equivalence is only required with respect to the permission types that are the subject of the 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 Jakarta 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 (global) `Policy` provider.

Each `policy` context contains all of the policy statements (as defined by this specification) that effect access to the resources[2] 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 (Setting the Policy Context).

3.1.1.1. Policy Context Life Cycle

Figure 3-2 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`.

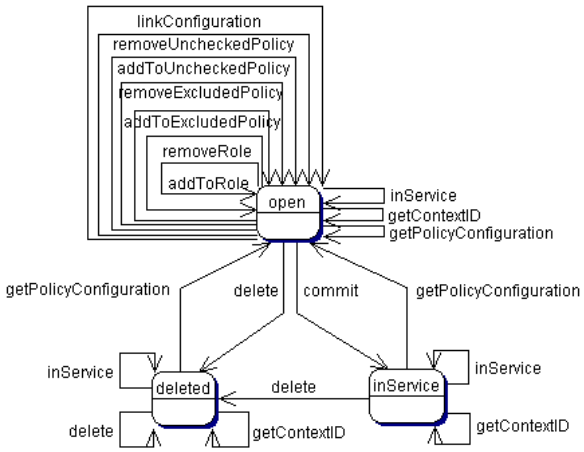

Figure 3-2 PolicyConfiguration State Table

3.1.1.2. Linking Policy Contexts

In the Jakarta 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 Servlet Policy Context Identifiers. 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. Before the translation is performed, all policy statements must have been removed[3] from the policy context associated with the returned PolicyConfiguration.

3.1.3.1. Programmatic Servlet Registrations

Jakarta Servlet containers support the programmatic registration and security configuration of servlets. The servlet policy translation defined by this subcontract is described assuming that all such programmatic registration and security configuration has completed (for the servlet module corresponding to the policy context) before the translation is performed and that the resulting security related configuration has been represented in its equivalent form within the deployment descriptors on which the translation is performed. Where this is not the case, the result of the translation must be equivalent, as described previously, to the translation that would occur if it was the case. The mapping to equivalent deployment descriptor representation of security related configuration corresponding to programmatically registered servlets is defined in the Jakarta Servlet specification.

If the results of a prior translation are invalidated by subsequent programmatic registration and security configuration (as might occur if an initial translation is performed before the programmatic effects), the translation must be repeated. Before the translation is repeated, a reference must be obtained to the PolicyConfiguration object in the open state, and its policy statements must be removed. If the PolicyConfiguration has already been linked to other committed policy contexts, then it may be necessary or preferable (in order to satisfy the linking requirements defined in Deploying an Application or Module) to obtain the reference and remove the policy statements while preserving the linkages established for the context by the prior translation. Policy statements may be removed while preserving linkages by calling the removeUncheckedPolicy, removeExcludedPolicy, and removeRole methods on the open PolicyConfiguration object.

3.1.3.2. 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[4] 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 obtained by combining (as defined in "Combining HTTP Methods") the collections containing the pattern and occurring in a constraint with an excluding auth-constraint. The

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. If the “any authenticated user” role-name, “**”, occurs in an auth-constraint, a WebResourcePermission must also be added to the “**” role. 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; which must not include the role “**” unless the application has defined an application role named “**”. Each WebResourcePermission object must be constructed using the qualified pattern as its name and with actions defined by combining (as defined in "Combining HTTP Methods") 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 combining (as defined in "Combining HTTP Methods") 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[5] a representation of the acceptable connection type to the HTTP method specification obtained by combining (as defined in "Combining HTTP Methods") 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 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 obtained 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 possible HTTP method value[6]. The permission objects must be constructed using the qualified pattern as their name and with actions represented by an HTTP method specification that identifies all of the HTTP methods that do not occur in combination with the pattern. The form of the HTTP method specification used in the permission construction depends on the representation of the methods that occurred in combination with the pattern. If the methods that occurred are represented by an HttpMethodExceptionList as defined in "HTTP Method Exception List"), the permissions must be constructed using an HTTPMethodList naming all of the HTTP methods named in the exception list. Conversely, if the methods that occurred are represented by an HTTPMethodList, the permissions must be constructed using an HTTPMethodExceptionList naming all of the HTTP methods that occurred with the pattern. If a deny uncovered HTTP methods semantic is in effect for the web

module associated with the `PolicyContext`, the resulting permissions must be added to the excluded policy statements by calling the `addToExcludedPolicy` method on the `PolicyConfiguration` object. Otherwise, the 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 Jakarta 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 Jakarta 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 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[7] be dropped from the list.

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

QualifiedPattern ::= Pattern QualifyingPatternList
```

All colon characters occurring within Pattern and QualifyingPattern elements must be transformed to escaped encoding[8] prior to inclusion of the corresponding element in the QualifiedPattern.

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.

Combining HTTP Methods

The section defines the rules for combining HTTP method names occurring in web-resource-collection elements that apply to a common url-pattern. The rules are commutative and associative and are as follows:

- Lists of http-method elements combine to yield a list of http-method elements containing the union (without duplicates) of the http-method elements that occur in the individual lists.
- Lists of http-method-omission elements combine to yield a list containing only the http-method-omission elements that occur in all of the individual lists (i.e., the intersection).
- A list of http-method-omission elements combines with a list of http-method elements to yield the list of http-method-omission elements minus any elements whose method name occurs in the http-method list.
- An empty list (of http-method and http-method-omission elements) represents the set of all possible HTTP methods, including when it results from combination according to the rules described in this section. An empty list combines with any other list to yield the empty list.

When these combining rules are applied to a list of collections, the result is always either an empty list, a non-empty list of http-method elements, or a non-empty list of http-method-omission elements. When the result is an empty list, the corresponding actions value is the null (or the empty string) value. When the result is a non-empty list of http-method elements the corresponding actions value is a comma separated list of the HTTP method names occurring in the http-method elements of the list. When the result is a non-empty list of http-method-omission elements the corresponding actions value is an HTTP method exception list (as defined in "HTTP Method Exception List") containing the HTTP method names occurring in the http-method-omission elements of the list. The following table contains the three alternative combination results and their corresponding actions values.

Table 3-1 HTTP Method Combination to Actions Correspondence

Combination Result	Actions Value
empty list	null or empty string
list of http-method elements	HttpMethodList (e.g., “GET,POST”)
list of http-method-omission elements	HttpMethodExceptionList (e.g.,”!PUT,DELETE”)

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 represents 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 exclamation point (i.e., “!”) character. A comma separated list of one or more HTTP method names must follow the exclamation 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
```

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 3-2 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

3.1.3.3. 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. If the “any authenticated user” `role-name`, “**”, does not appear in a `security-role-ref` within the servlet, a `WebRoleRefPermission` must also be added for it. 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 and the “any authenticated user” role, “**”, an additional `WebRoleRefPermission` must[9] 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.

3.1.3.4. 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 3-3 url-pattern Types by Example

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

3.1.3.5. 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.

```
<!--
    The following security-constraint excludes access to the patterns and method
    combinations defined by the two contained web-resource-collections.

    The first collection excludes access
    by all methods except GET and POST, while the second collection excludes
    access by all 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-omission>GET</http-method-omission>
            <http-method-omission>POST</http-method-omission>
    </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 patterns 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>
```

Qualified URL Pattern Names from Example 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 Qualified URL Pattern Names from Example 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 3-4 Qualified URL Pattern Names from Example

Qualified URL Pattern Name type	Canonical Form
/a	"/a
/b	/b
pass:[/a/*:/a]	pass:[/a/*:/a]
pass:[/b/*:/b]	pass:[/b/*:/b]
pass:[/b/*:/b]	pass:[/b/*:/b]
pass:[*.asp:/a/*:/b/*]	pass:[*.asp:/a/*:/b/*]
pass:[/:/a:/b:/a/*:/b/*:*.asp]	pass:[/:/a/*:/b/*:*.asp]

Permissions and PolicyConfiguration Operations from Example 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 Qualified URL Pattern Names from Example. As noted previously, any equivalent form of the qualified names, including their canonical forms, could have been used in the permission constructions.

Table 3-5 Permissions and PolicyConfiguration Operations from Example

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

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

Regarding the `null` in the third column of Permissions and PolicyConfiguration Operations from Example; the canonical form for the set of all HTTP Methods (including all extension methods) is null.

3.1.4. 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. In Jakarta Servlet containers, the policy statements resulting from the deployment and initialization of a web module, must represent the effects of any programmatic registration and security configuration of servlets that occurred during the initialization of the module.

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[10] by calling the linkConfiguration method on the `PolicyConfiguration` objects of the provider.

After the translation and linking has occurred (note that they may occur in either order) for a policy context, the `commit` method must be called on the corresponding `PolicyConfiguration` object to place it in service such that its policy statements will be assimilated by the corresponding `Policy` providers. These three operations, translate, link and commit, must be performed for all of the policy contexts of the application.

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. If a module corresponding to a policy context may have inter-module, initialization-time, dependencies that must be satisfied before the translation of the policy context of the dependent module can be completed[11], the `commit` of the depended upon modules must occur before the initialization of the dependent module, and the calls to `Policy.refresh` described above must additionally occur after the processing of the depended upon modules and before the initialization of the dependent module.

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 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.

3.1.5. 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.

3.1.6. 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.

In Jakarta Servlet containers, if any programmatic registration and security configuration of servlets has occurred during the initialization of a web module associated with a pre-exiting policy context, the corresponding `PolicyConfiguration` object must be opened, its policy statements must be removed, and the policy translation of the module must be repeated to include the programmatic effects. The `PolicyConfiguration` object must be committed, and an additional call to `Policy.refresh` must be made after all such `PolicyConfiguration` objects are committed.

3.1.7. 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 in Deploying an Application or

Module, 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 and linkages 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 and linkages (as necessary), the deployment tools must proceed as described in Deploying an Application or Module.

3.2. What the Provider Must Do

If a provider wants to replace the default `PolicyConfiguration` of a Jakarta Authorization implementation, the provider must include an implementation of the `jakarta.security.jacc.PolicyConfigurationFactory` class along with a matched implementation of a class that implements the `jakarta.security.jacc.PolicyConfiguration` interface.

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. For the any "authenticated user role", "*", and unless an application specific mapping has been established for this role, the provider must ensure that all permissions added to the role are granted to any authenticated user. 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.

[1] For some permission types it will generally not be possible to use the implies method of the `PermissionCollection` to compute collection equivalence (because the implies method is unable to determine when a collection contains all the permissions implied by a wild carded form of the permission).

[2] An exception to this rule is described in Jakarta Enterprise Beans Policy Context Identifiers.

[3] This can be achieved by passing true as the second parameter in the call to `getPolicyConfiguration`, or by calling delete on the `PolicyConfiguration` before calling `getPolicyConfiguration` to transition it to the open state.

[4] 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

[5] The value null should be used as the actions value in the construction of a `WebUserDataPermission` when both the HTTP method specification, 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.

[6] The set of all possible HTTP methods is non-enumerable and contains the traditional HTTP methods (i.e., 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". A null or the emptyString HTTP method specification is used to this set.

[7] 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 `URLPatternSpec` 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/*)

[8] See Colons Within path-segment of Request URI for details.

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

[10] 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, or that all

contexts must be linked before any can be committed.

[11] Such as having a Jakarta Servlet `ServletContextListener` configured that could programmatically register a servlet and configure its security constraints.

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 Jakarta EE containers.

4.1. Policy Enforcement by Servlet Containers

Jakarta 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. All colon characters occurring within the name must be represented using escaped encoding[1]

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 Jakarta Servlet container must obtain a `WebUserDataPermission` object with name obtained from the request URI as defined in 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 Jakarta Servlet container must use one of the methods described in Checking 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 Jakarta Servlet Specification. If access is permitted, the

request must be subjected to a pre-dispatch decision.

4.1.3. Pre-dispatch Decision

The Jakarta Servlet container must obtain a `WebResourcePermission` object with name obtained from the request URI as defined in Permission Names for Transport and Pre-Dispatch Decisions. The actions of the obtained permission must be the HTTP method of the request. The Jakarta Servlet container must use one of the methods described in 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 Jakarta 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 Jakarta Servlet Specification.

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 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 policy statements of the policy context identified by calling `PolicyContext.getContextID` to determine if they imply the permission being checked (called the "checked permission" hence forward).

A Policy implementation can obtain all the policy statements (as `Permission` instances) by calling `PolicyConfigurationFactory.getPolicyConfiguration()`:

```
PolicyConfiguration policyConfiguration =
    PolicyConfigurationFactory.get().getPolicyConfiguration();
```

`getPolicyConfiguration()` is a convenience method that internally uses the current policy context as

obtained from `PolicyContext.getContextID`. It is equivalent to:

```
PolicyConfiguration policyConfiguration =
    PolicyConfigurationFactory.get().getPolicyConfiguration(PolicyContext.
getContextID());
```

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.

For example, in `jakarta.security.jacc.Policy.implies(Permission, Subject)`:

```
boolean isExcluded =
    PolicyConfigurationFactory.get()
                              .getPolicyConfiguration()
                              .getExcludedPermissions()
                              .implies(permissionToBeChecked);

if (isExcluded) {
    return false;
}
```

Otherwise, if one or more unchecked policy statements imply the checked permission, the checked permission must be determined to be granted.

For example, in `jakarta.security.jacc.Policy.implies(Permission, Subject)`:

```
boolean isUnchecked =
    PolicyConfigurationFactory.get()
                              .getPolicyConfiguration()
                              .getUncheckedPermissions()
                              .implies(permissionToBeChecked);

if (isUnchecked) {
    return true;
}
```

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 caller principal in the `Subject` or set of principals 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 caller principal in the `Subject` or set of principals.

One way of determining this is to obtain the roles associated with the caller principal, and checking these against the `perRolePermissions`. Roles associated with the caller principal can be obtained from the `Subject` or set of principals using the `PrincipalMapper`.

For example, in `jakarta.security.jacc.Policy.implies(Permission, Subject)`:

```
PrincipalMapper principalMapper = PolicyContext.get(PRINCIPAL_MAPPER);
Set<String> callerRoles =
    principalMapper.getMappedRoles(subject);

Map<String, PermissionCollection> perRolePermissions =
    PolicyConfigurationFactory.get()
                              .getPolicyConfiguration()
                              .getPerRolePermissions();

for (String role : callerRoles) {
    if (perRolePermissions.containsKey(role) &&
        perRolePermissions.get(role).implies(permissionToBeChecked)) {
        return true;
    }
}

return false;
```

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 Jakarta Authorization `Policy` interface contains a default method that captures the above described rules:

```
default boolean implies(Permission permissionToBeChecked, Subject subject) {
    if (isExcluded(permissionToBeChecked)) {
        return false;
    }

    if (isUnchecked(permissionToBeChecked)) {
        return true;
    }

    return impliesByRole(permissionToBeChecked, subject);
}
```

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 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 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[2] 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 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 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.

- 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 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 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. Component runAs Identity

The identity used by Jakarta Servlet 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 Jakarta 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.

4.4. 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 Jakarta Servlet 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.

4.4.1. 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 must19 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 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). When a required context handler for which registration has been delayed is invoked, the container may return null, and must complete the registration of the handler before returning.

A provider that is dependent on a handler, should force registration of the handler in advance of the provider's processing of a component request for which the handler is required. This can be accomplished by invoking the required handler during initialization of the provider.

4.4.1.1. Container Subject Policy Context Handler

All Jakarta 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 a Jakarta Servlet component calls `isUserInRole`). In all cases, if the identity of the corresponding `Subject` has not been established or authenticated, this handler must return the value null.

4.4.1.2. HttpServletRequest Policy Context Handler

All Jakarta Servlet containers must register a `PolicyContextHandler` whose `getContext` method returns a `jakarta.servlet.http.HttpServletRequest` object when invoked with the key "jakarta.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.

4.5. Checking 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. The `WebUserDataPermission` policy statements resulting from the translation of Jakarta 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.

- The container calls `Policy.implies` with two arguments; the permission being checked and a `Subject` that need not be constructed with principals. The checked permission is granted if `Policy.implies` returns true. Otherwise, the permission is not granted.
- The container calls one of the overloaded methods of `Policy.implies`, which are provided for convenience (see their javadoc for details). likewise, the checked permission is granted if the overloaded `Policy.implies` returns true. Otherwise, the permission is not granted.
- The container calls `Policy.getPermissionCollection` with a `Subject` 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.

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

4.6. 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 a `Subject` containing the principals of (only) the caller[3]. 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.

- container calls `Policy.implies` with two arguments; the permission being checked and a `Subject`

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 container calls `Policy.getPermissions` with an argument `Subject` 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[4]

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

4.7. 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.

4.8. Optimization of Permission Evaluations

Jakarta Authorization implementations 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 Jakarta Authorization implementation's threshold for being effected by policy changes:

- Jakarta Authorization implementations may reuse an authorization result obtained from a previous equivalent permission evaluation.
- Jakarta Authorization implementations may reuse an authorization result obtained for an unauthenticated caller (i.e. a caller with no principals) performed as defined in 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 Jakarta Authorization implementations determines when a repeated evaluation will return the same result. That said, one way that Jakarta Authorization implementations 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 Jakarta Authorization implementations to receive such notification could be for them to register to the `"java.security.Policy.supportsReuse"` key a `PolicyContextHandler` and for the Jakarta Authorization implementation 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.

[1] The `HttpServletRequest` based constructors of `WebResourcePermission` and `WebUserDataPermission` must perform the escaped ecoding. For all other constructors, the encoding must be performed prior to invoking the constructor. See issue Section B.22, “Colons Within path-segment of Request URI.

[2] The syntax and semantics of an `HTTPMethodExceptionList` are described in a subsection of Translating security-constraint Elements

[3] 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.

[4] 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.

Chapter 5. Support for legacy technologies

The Support for legacy technologies defines the support for technologies that are considered legacy or de-emphasized in Jakarta EE. The technologies and their support documented here may become deprecated and ultimately pruned in the future. As of now they are fully supported.

5.1. Policy Configuration Subcontract for Jakarta Enterprise Beans

5.1.1. Assumptions

1. Where the Jakarta Enterprise Beans is incomplete or ambiguous in its 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 committers of the corresponding component specification.
2. The Jakarta Enterprise Beans policy decisions performed by providers may require access to the arguments of the Enterprise Bean invocation and or (for entity beans) the container managed Enterprise Bean instance state.

5.1.2. What a Jakarta EE Platform's Deployment Tools Must Do

In addition to What a Jakarta EE Platform's Deployment Tools Must Do the rules for combining authorization policy annotations with declarative policy statements are described in the Jakarta Enterprise Beans, Jakarta Servlet, and Jakarta EE platform specifications.

5.1.3. Jakarta Enterprise Beans Policy Context Identifiers

To satisfy this specification, an application server must establish Jakarta Enterprise Beans policy context identifiers sufficient to differentiate all instances of the deployment of an Jakarta Enterprise Beans 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 Jakarta Enterprise Beans jars, no two jars that share at least one `ejb-name` value in common may share the same policy context identifiers.

In cases where Jakarta Enterprise Beans may be packaged in war files, the application server must assign the Jakarta Enterprise Beans to a policy context distinct from that to which any web components are assigned.

The policy contexts assigned to web applications and web modules must be distinct from those to which any Jakarta Enterprise Beans components are assigned.

5.1.4. Translating Jakarta Enterprise Beans 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 Jakarta Enterprise Beans Policy Context Identifiers. 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 Jakarta Enterprise Beans method for which the container performs pre-dispatch access decisions is implied by at least one permission resulting from the translation. Before the translation is performed, all policy statements must have been removed[1] from the policy context associated with the returned `PolicyConfiguration`.

5.1.4.1. Translating Jakarta Enterprise Beans 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. These `addToRole` calls must be made for any `role-name` used in the `method-permision` which may include the role-name “**”; which, by default, is mapped to any authenticated user.

5.1.4.2. Translating the Jakarta Enterprise Beans 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 Translating Jakarta Enterprise Beans 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.

5.1.4.3. Translating Jakarta Enterprise Beans security-role-ref Elements

For each `security-role-ref` element appearing in the deployment descriptor, a corresponding `EJBRoleRefPermission` must be created. The value of the `ejb-name` element within the element containing the `security-role-ref` element must be used as the name of the `EJBRoleRefPermission`. 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`.

Additional `EJBRoleRefPermission` objects must be added to the `PolicyConfiguration` as follows. For

each element in the deployment descriptor for which the Jakarta Enterprise Beans descriptor schema supports[2] inclusion of `security-role-ref` elements, an EJBRoleRefPermission 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 element. If the “any authenticated user” `role-name` , “**”, does not appear in a `security-role-ref` within the element, a `EJBRoleRefPermission` must also be added for it. The name of each such `EJBRoleRefPermission` must be the value of the `ejb-name` element within the element in which the `security-role-ref` elements could otherwise occur. The actions (that is, reference) of each such `EJBRoleRefPermission` must be the corresponding (non-appearing) `role-name` . The resulting permissions must be added[3] to the corresponding roles by calling the `addToRole` method on the `PolicyConfiguration` object.

5.2. Policy Decision and Enforcement Subcontract for Jakarta Enterprise Beans

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

5.2.1. Jakarta Enterprise Beans Pre-dispatch Decision

The Jakarta Enterprise Beans 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 akarta Enterprise Bean by identifying the method interface, method name, and method signature as defined for a methodSpec in the documentation of the `EJBMethodPermission` class.

The Jakarta Enterprise Beans container must use one of the methods described in 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 Jakarta Enterprise Beans container may only dispatch the request to the Jakarta Enterprise Bean resource, if the `EJBMethodPermission` is determined to be granted to the caller. Otherwise the request must be rejected with the appropriate exception, as defined by the corresponding Jakarta Enterprise Beans specification.

5.2.2. Jakarta Enterprise Beans Application Embedded Privilege Test

When a Jakarta Enterprise Bean 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 Jakarta Enterprise Bean 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 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`.

5.3. Provider Support for Jakarta Enterprise Beans Policy Enforcement

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

5.3.1. Jakarta Enterprise Beans Policy Decision Semantics

A Policy provider must employ the policy decision semantics described in Servlet Policy Decision Semantics in the processing of Jakarta Enterprise Beans 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.

5.3.1.1. 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.

EJBMethodPermission methodSpec Matching Examples demonstrate the properties of `EJBMethodPermission` matching by example.

Table 5-6 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

type	methodInterface Spec	methodName Spec	methodParams Spec	implies checked permission
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

5.3.1.2. 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.

5.3.2. Component runAs Identity

The identity used by Jakarta Enterprise Beans 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 Jakarta Enterprise Beans 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.

5.3.3. 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 Jakarta Enterprise Beans 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.

5.3.4. Policy Context Handlers

Following the requirements of the top level section Policy Context Handlers, Jakarta Enterprise Beans containers have the specific requirements as detailed in the following sub-sections.

5.3.4.1. Container Subject Policy Context Handler

All Jakarta Enterprise Beans 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 a Jakarta Enterprise Beans 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.

5.3.4.2. SOAPMessage Policy Context Handler

All Jakarta Enterprise Beans containers must register a `PolicyContextHandler` whose `getContext` method returns a `jakarta.xml.soap.SOAPMessage` object when invoked with the key "jakarta.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.

5.3.4.3. EnterpriseBean Policy Context Handler

All Jakarta Enterprise Beans containers must register a `PolicyContextHandler` whose `getContext` method returns a `jakarta.ejb.EnterpriseBean` object when invoked with the key "jakarta.ejb.EnterpriseBean". When this handler is activated, the container must return the `EnterpriseBean` object corresponding to the Jakarta Enterprise Beans 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 Jakarta Enterprise Beans `Remote`, `Local`, or `ServiceEndpoint` interfaces of the `EnterpriseBean` object. The value null must be returned if the bean implementation class does not implement the `jakarta.ejb.EnterpriseBean` interface.

5.3.4.4. Jakarta Enterprise Beans Arguments Policy Context Handler

All Jakarta Enterprise Beans containers must register a `PolicyContextHandler` whose `getContext` method returns an array of objects (`Object[]`) containing the arguments of the Jakarta Enterprise

Beans method invocation (in the same order as they appear in the method signature) when invoked with the key “jakarta.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 Jakarta Enterprise Beans 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]`).

5.3.5. Checking Grants

As described in Checking Grants.

5.3.6. Checking the Caller for a Permission

As described in Checking the Caller for a Permission

[1] This can be achieved by passing `true` as the second parameter in the call to `getPolicyConfiguration`, or by calling `delete` on the `PolicyConfiguration` before calling `getPolicyConfiguration` to transition it to the open state.

[2] Jakarta Enterprise Beans supports inclusion of `security-role-ref` elements in entity and session elements. Future versions could support inclusion in `message-driven`.

[3] For example, if an application declares roles {R1, R2, R3} and defines a session Jakarta Enterprise Bean named “shoppingCart” that contains one `security-role-ref` element with `role-name` R1, then an additional `EJBRoleRefPermission` must be added to each of the roles R2 and R3. The name of both permissions must be “shoppingCart”, and the actions value of the permission added to role R2 must be “R2”, and the actions value of the permission added to role R3 must be “R3”.

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, Harvard University, March 1997, [Keywords]

Jakarta EE 11 Specification [Jakarta EE 11 Specification], available at: https://jakarta.ee/specifications/platform/11/

Jakarta Servlet Specification, Version 6.1 [Jakarta Servlet Specification], available at: https://jakarta.ee/specifications/servlet/6.1/

Jakarta Enterprise Beans, Version 4.0 [Jakarta Enterprise Beans Specification], available at: _https://jakarta.ee/specifications/enterprise-beans/4.0/

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 (Partial) - 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 Jakarta Servlet.

B.5. Evolution of Deployment Policy Language

The Policy Configuration 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 Jakarta Enterprise Beans and Jakarta 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 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 Jakarta Servlet Constraint Matching Semantics

The definition of the `security-constraint` matching and enforcement semantics were under specified in the historical pre-Jakarta Servlet 2.2 and 2.3 specifications. The contract defined in this document has clarified these semantics; however there was an issue until these clarifications were incorporated into the pre-Jakarta Servlet.

Resolution - The pre-Jakarta Servlet 2.4 specification and onwards 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 Jakarta Enterprise Bean (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 Jakarta 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 a compatible implementation.

B.10. PolicyContext Identifiers are Unknown to Components

Although not strictly speaking within the scope of this specification, the work of this specification 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.11. 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 recommendation to Provider Configuration Subcontract] that the *javax.security.auth.SubjectDomainCombiner* of an application server must *combine* into the permission collection returned by *javax.security.auth.Policy.getPermisions*.

B.12. 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 NOT apply. 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.13. 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 `jakarta.security.jacc` package, and the methods of the classes and interfaces identified above were modified to throw this checked exception as appropriate.

B.14. 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.15. 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 pre-Jakarta Enterprise Beans 2.1.

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

B.16. 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 opposed to in the form returned by `Class.getName()` (i.e. `[LComponentType;`).

B.17. 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.18. Calling isUserInRole from JSP not mapped to a Servlet

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

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.19. Support for HTTP Extension Methods

Pre-Jakarta Servlet 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.20. 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 Jakarta 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 Jakarta Servlet clarification. Recommend that Jakarta Servlet standarize an `HttpServletRequest` attribute that can be used to portably obtain the `requestURI` following welcome file mapping. Once this attribute is standardized, The `HttpServletRequest` based constructors of `WebResourcePermission` and `WebUserDataPermission` would use its value to establish the permission name.

B.21. 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 - Require that containers use escaped encoding (as defined in RFC 2396) on colon characters occuring within `url-patterns` obtained from `web.xml`. Also require that containers encode colons occuring within patterns extracted from `HttpServletRequest` objects and used to create the names of checked `WebResourcePermission` and `WebUserDataPermission` objects. Also require the `HttpServletRequest` based constructors of `WebResourcePermission` and `WebUserDataPermission` apply escaped encoding to colons occuring in the names the derived from the request URI. Note that the colon character is represented as %3A in escaped encoding.

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.

1. Specification title changed to J2EETM Authorization Contract for Containers .Added additional definitions to the terminology section for [a90] and [a92].
2. 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.
3. Added an Assumption and a corresponding requirement with respect to support for Policy Providers that get all permissions before returning from Policy.
4. Clarified relationship to Servlet and EJB specification of authorization semantics.
5. Changed all references to “VM” to “JRE”
6. Changed all references to “deploy tool” to “deployment tool”
7. Removed empty brackets from all method names in prose.

C.2.2. Changes to Provider Configuration Subcontract

1. Rewrote replacability paradigm. New model does not require replacement of permission implementations.
2. 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

1. Changed to be compatible with the changes made (for replacability) to the Provider Configuration Subcontract.

C.2.4. Changes to Policy Decision Subcontract

1. 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.
2. Added clarification of the matching of an excluded policy statement to a granted permission.

3. Clarified the policy decision algorithms in Checking 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.
4. In Checking the Caller for a Permission added requirement that caller identity not include principals of any trusted (other than the caller).
5. 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.
6. Rewrote the a section 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

1. 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.
2. Specified exceptions to be thrown by methods of PolicyConfiguration interface.
3. 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

1. Resolved issue Principals Passed to Providers in Subjects
2. Resolved issue Permission Spanning in RoleRefPermission
3. Added new issue PolicyContext Identifiers are Unknown to Components

C.3. Changes in Public Draft Version 0.2

C.3.1. General

1. Specification title changed to JavaTM Authorization Contract for Containers
2. Corrected audience to be the public
3. 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

1. Replaced most references to container with JRE, as a provider integrates per JRE.

C.3.3. Changes to Policy Decision Subcontract

1. In Jakarta Enterprise Beans Policy Decision Semantics, replace references to “subject” with “access control context”.
2. In Component runAs Identity, softened requirement that container prevent component from

modifying its runAs identity by saying that this must be the default policy.

3. Changes references to “access exception” to AccessControlException.

C.3.4. Changes to Issues

1. Added Introductory paragraph.
2. Added new issue, 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

1. The license page was changed; most notably the license number.
2. Improper uses of the word “which” were replaced with the word “that”.
3. The word “shall” was replaced with the word “must”.
4. 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

1. The preface was changed to reflect the purpose of the PFD
2. the definition of hostname was modified so that hostnames are no longer required to be used in servlet policy context identifiers.
3. 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

1. PolicyContext Class and Context Handlers was inserted to describe the PolicyContext utility class and the PolicyContextHandler interface.
2. What the Jakarta Authorization Implementation 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

1. The examples in What a Jakarta EE Platform’s Deployment Tools Must Do, were modified to reflect changes to policy context identifiers and their removal from permission names.
2. The requirement that the names of checked permissions identify the policy context was removed from Policy Contexts and Policy Context Identifiers
3. The linkConfiguration method name replaced the incorrect link method name in Linking Policy Contexts

4. Servlet Policy Context Identifiers was moved to follow 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.
5. Translating security-constraint Elements was made a subsection of a new Translating Servlet Deployment Descriptors and changed to deal with the removal of policy context identifiers from permission names.
6. The part of the translation remaining in Translating security-constraint Elements was modified to yield an OR constraint combination semantic.
7. The description of the mapping of transport guarantees to unacceptable connection types was moved to a new "Mapping Transport Guarantee to Connection Type"
8. Translating Servlet security-role-ref Elements was modified to reflect the removal of policy context identifiers from permission names.
9. Translating Jakarta Enterprise Beans method-permission Elements was made a subsection of a new Translating Jakarta Enterprise Beans Deployment Descriptors and changed to deal with the removal of policy context identifiers from permission names.
10. A new section Jakarta Enterprise Beans Policy Context Identifiers, was added to describe the selection of EJB policy context identifiers.
11. Translating Jakarta Enterprise Beans method-permission Elements, Translating the Jakarta Enterprise Beans exclude-list, and Translating Jakarta Enterprise Beans security-role-ref Elements were all changed to reflect the removal of the policy context identifier from permission names.
12. Undeploying an Application or Module was modified to reflect the use of the PolicyContext class to define the policy context.
13. a549 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

1. The name was changed to the “Policy Decision and Enforcement Contract”.
2. Evaluation of Transport Guarantees, Pre-dispatch Decision, and Application Embedded Privilege Test were changed to reflect the removal of the policy context identifier from permission names.
3. Section 4.2.1 “Servlet Constraint Matching Semantics”, was replaced by two sections; Servlet Policy Decision Semantics, and WebResourcePermission Matching Rules.
4. The latter describes the processing of servlet constraints in a manner related to the three types of policy statements created via the PolicyConfiguration interface.
5. Section 4.2.2.1, “Servlet URL-Pattern Matching Rules” was renamed.
6. 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.
7. Section 4.2.3, “WebRoleRefPermission Processing Semantics” was added as the public draft mistakenly assumed that the Servlet policy model was just about constraints.

8. Jakarta Enterprise Beans Pre-dispatch Decision and Jakarta Enterprise Beans Application Embedded Privilege Test were changed to reflect the removal of the policy context identifier from permission names.
9. A new 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.
10. A new 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: Container Subject Policy Context Handler, SOAPMessage Policy Context Handler, HttpServletRequest Policy Context Handler, EnterpriseBean Policy Context Handler, and Jakarta Enterprise Beans Arguments Policy Context Handler.
11. The methods for checking policy as defined in Checking 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.
12. The same changes as described in the previous change item were applied to Checking the Caller for a Permission.
13. 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.
14. A new section was introduced to describe requirements for chaining policy evaluation through to the provider of the default policy context.

C.4.6. Changes to API

1. 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.
2. Added the PolicyContextHandler interface.
3. Removed the special purpose, EntityBean and
4. Argument array constructors from the EJBMethodPermission class.
5. Removed the special purpose, EntityBean constructor from the EJBRoleRefPermission class.
6. 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.
7. In the PolicyConfiguration interface, changed the name of the getPolicyContextId method to getContextID.
8. 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.

9. 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

1. Changed the introductory material to indicate that all of the issues have been resolved.
2. The resolution of Issue Configuration of Permissions with Parameters, was changed to reflect the introduction of policy context handlers.
3. Issue 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.
4. Clarification of Jakarta Servlet Constraint Matching Semantics, was resolved with the rewrite of Section WebResourcePermission Matching Rules, and with the expectation that the Servlet EG will adopt a change to section SRV.12.8 of the Servlet specification.
5. Issue References and Arguments in EJBMethodPermisison, was resolved with the introduction of policy context handlers.
6. Issue "Integrating Principal-to-Role Mapping with the Deployer Console", was made optional functionality.
7. 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

1. The restriction that entities be identified by principal was removed from the definition of grant.

C.5.2. Changes to Policy Configuration Subcontract

1. In What a Jakarta EE Platform's Deployment Tools Must Do, the argument to linkConfiguration was corrected in the example.

C.5.3. Changes to Policy Decision Subcontract

1. Section 4.2.2.1, "Servlet URL-Pattern Matching Rules" was modified to indicate that pattern length only is significant among path prefix matches.
2. 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".
3. Section 4.2.3, "WebRoleRefPermission Processing Semantics" was added as the public draft mistakenly assumed that the Servlet policy model was just about constraints.
4. Jakarta Enterprise Beans Pre-dispatch Decision and Application Embedded Privilege Test were changed to reflect the removal of the policy context identifier from permission names.

C.5.4. Changes to History

1. 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

1. 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

1. A clarifying sentence was added to the end of 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

1. Section 4.2.3, “WebRoleRefPermission Processing Semantics” was simplified, as much of its content was not pertinent to the WebRoleRefPermission class.
2. 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.
3. In 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.
4. In EnterpriseBean Policy Context Handler, the handler return type was corrected.

C.6.4. Changes to API

1. 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

1. 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

1. Updated Status section
2. Acknowledged all contributors, including RI and TCK team, and all those who commented on the specification.

C.7.3. Changes to Overview

1. Added dashed lines to Figure 1-1 to represent PolicyContext interactions.
2. Modified requirement 7, to reflect change is treatment of permissions derived from security-constraints.

C.7.4. Changes to Provider Configuration Subcontract

1. Added two new sentences to the end of 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.
2. In What the Jakarta Authorization Implementation Must Do, added all elements of the jacc package to the list of things that an application server must bundle.
3. In What the Jakarta Authorization Implementation 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.
4. In What the Jakarta Authorization Implementation 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

1. In the examples in What a Jakarta 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).
2. In the examples in What a Jakarta EE Platform's Deployment Tools Must Do, a new stanza was added to place the policy context in service.
3. Policy Context Life Cycle, was added.
4. In 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.
5. 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.
6. 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.
7. 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". transport-guarantee to Acceptable Connection Mapping 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.

8. Servlet URL-Pattern Matching Rules, was added to support the pattern qualification section, and relevant sections of the enforcement subcontract.
9. Example was added
10. In Translating Jakarta Enterprise Beans Deployment Descriptors, the call to getPolicyConfiguration was augmented with a second parameter to ensure that all policy statements are removed from the context.
11. The last paragraph of Translating the Jakarta Enterprise Beans exclude-list, was clarified.
12. Deploying an Application or Module, Undeploying an Application or Module, Deploying to an existing Policy Configuration, and Redeploying a Module, were all changed o reflect the introduction of the policy context life cycle and the commit method.
13. The inService method was added to the factory methods called out in the first paragraph of a549, 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

1. Policy Enforcement by Servlet Containers, was modified to require that containers use Policy to make access control decisions.
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”.
3. Evaluation of Transport Guarantees, and Pre-dispatch Decision, were changed to reference the error processing defined in the Servlet specification.
4. Servlet Policy Decision Semantics, was rewritten to reflect the qualification of the permission names, and the change to conventional permission evaluation semantics.
5. WebResourcePermission Matching Rules, WebRoleRefPermission Matching Rules, and WebUserDataPermission Matching Rules were added to define the permission specific matching semantics necessary to support the policy decision semantics.
6. 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.
7. Policy Decision and Enforcement Subcontract for Jakarta Enterprise Beans, was modified to require that containers use Policy to make access control decisions.
8. Jakarta Enterprise Beans Policy Decision Semantics, was replaced with a simplified section that references Servlet Policy Decision Semantics.
9. EJBMethodPermission Matching Rules, and 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”.

10. The last paragraph of Component runAs Identity, was modified to ensure that the AccessControlContext includes a SubjectDomainCombiner.
11. In 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".
12. In Container Subject Policy Context Handler replaced "caller's identify" with "caller's identity".
13. In SOAPMessage Policy Context Handler, reduce to only EJB container, and added additional qualification of the request coming in at the ServiceEndpoint method interface.
14. In Jakarta Enterprise Beans 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.
15. Renamed section Checking Grants and changed it to reflect the changes made to WebUserDataPermissions such that they are no longer "excluded" permissions.
16. In <<a745>, changed replaced contains with inService method.

C.7.7. Changes to API

1. A new class diagram was imported to reflect the changes to the API, most notably the introduction of the PolicyContextException class.
2. The javadocs were regenerated to conceal implementation specific private instance variables.
3. Added "ServiceEndpoint" to the list of alternative MethodInterface identifiers for EJBMethodPermissions.
4. More completely specified EJBMethodPermission matching of methodNameSpec in implies
5. Added policy context life cycle, including description, and state table to PolicyConfiguration interface.
6. Added new methods "commit" and inService to the PolicyConfiguration interface.
7. Changed all the method signatures of the PolicyConfiguration interface to throw PolicyContextException, and described the other exceptions that implementations are required to throw.
8. Changed the documentation of getPolicyConfigurationFactory to properly identify the system property.
9. 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.
10. Renamed contains of PolicyConfigurationFactory class to inService.
11. Changed all the method signatures of the PolicyConfigurationFactory class to throw PolicyContextException, and described the other exceptions that implementations are required to throw.
12. Changed authorization requirement of the PolicyContext class to allow containers to be responsible for deciding how callers of this method must be authorized.
13. 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.

14. 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.
15. Modified the specification of the implies and equals methods of WebResourcePermission to account for the URLPatternSpec.
16. 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.
17. Changed BNF for “actions” of WebUserDataPermission such that a separating “:” is not required if a transportType is not explicitly specified.
18. Replaced transportTypeList in actions of WebuserdataPermission with a single transportType value.
19. Modified the specification of the implies and equals methods of WebUserDataPermission to account for the URLPatternSpec.
20. Comparable Interface was removed from WebResourcePermission and WebUserDataPermission.
21. 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.
22. In WebUserDataPermission constructor removed extra “and” in “…by calling and HttpServletRequest.isSecure()”.
23. In description of PolicyContextHandler.getContext, removed extra “the” from “and obtain from it the the”.

C.7.8. Changes to References

1. 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

1. Added new issue, Implementing Web Security Constraints as Permission.
2. Added new issue, Exception Handling.
3. Added new issue, PolicyConfiguration Commit.
4. Added new issue, Support for ServiceEndpoint methodInterface.

C.8. Changes in Proposed Final Draft 2 Expert Draft 2

C.8.1. Changes to Preface

1. fixed typos, and added additional RI team member to credits.

C.8.2. Changes to Policy Configuration Subcontract

1. In Servlet URL-Pattern Matching Rules, added additional clause to support universal matching by "/*".
2. In Example, Added comments to security-constraint elements, Also corrected qualified URL Pattern Names occurring in Qualified URL Pattern Names from Example and Permissions and PolicyConfiguration Operations from Example.
3. In 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

1. In Evaluation of Transport Guarantees, and 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".
2. Modified 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.
3. Added a new Matching Qualified URL Pattern Names to describe URLPatternSpec matching, and replaced the duplicate descriptions of this processing in sections WebResourcePermission Matching Rules and 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.
4. Added requirement that the comparisons defined by WebResourcePermission Matching Rules, WebRoleRefPermission Matching Rules, WebUserDataPermission Matching Rules, EJBMethodPermission Matching Rules, and EJBRoleRefPermission Matching Rules be case sensitive.
5. The word "form" was changed to "from" in first paragraph of Checking Grants.
6. In bullets 4 and 5 of Checking Grants, removed "that was constructed without static permissions and".
7. Rewrote a747 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

1. 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.
2. 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.

3. 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

1. Added a new first paragraph to 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.
2. 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).
3. In 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.
4. Added a new column to Qualified URL Pattern Names from Example of Example to represent the canonical form of the qualified names.The description of Permissions and PolicyConfiguration Operations from Example was modified to indicate that the names in its second column were obtained from the first column of Qualified URL Pattern Names from Example, and that any equivalent form of the qualified names, including their canonical forms, could have been used in the permission constructions.

C.9.2. Changes to Policy Decision and Enforcement Subcontract

1. In Evaluation of Transport Guarantees, clarified the actions value used for a request that arrives on an unprotected connection.

C.9.3. Changes to API

1. 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.
2. 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

1. 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.

2. 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

1. License was replaced

C.11.2. Changes to the Preface

1. The preface was changed to reflect the purpose of the Final Release.
2. Additional contributor names were added.

C.11.3. Changes to Overview

1. Added requirement to support 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

1. Added another catch clause to the code sample in What the Jakarta Authorization Implementation Must Do, to support verification that the loaded object is an instanceof javax.security.Policy.

C.11.5. Changes to Policy Configuration Subcontract

1. Added definition of what it means for two translations to be “equivalent” to What a Jakarta EE Platform’s Deployment Tools Must Do.
2. Added clarification to Translating security-constraint Elements to allow for “equivalent” translations.
3. Restated the translation description of Translating security-constraint Elements, such that it no longer prescribes the number of permissions that must be constructed.
4. Modified the title of the second column of transport-guarantee to Acceptable Connection Mapping.
5. Restated the translation description of 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.
6. Fixed a syntax problem, missing "<" in "urlPattern>", in Example.
7. Changed some of the actions values of Permissions and PolicyConfiguration Operations from Example, such that they are all in canonical form. Added table footnote to that effect.
8. Added clarification to Translating Jakarta Enterprise Beans method-permission Elements to

allow for “equivalent” translations.

9. Restated the translation description of Translating Jakarta Enterprise Beans method-permission Elements, such that it no longer such that it no longer prescribes the number of permissions that must be constructed.
10. Clarified the linking requirements of Deploying an Application or Module and of Redeploying a Module.
11. In Undeploying an Application or Module, Deploying to an existing Policy Configuration, and in Redeploying a Module, changed “must stop accepting” to “must stop dispatching” requests.

C.11.6. Changes to Policy Decision and Enforcement Contract

1. Added special rule for checking "/" to Evaluation of Transport Guarantees, and Pre-dispatch Decision.
2. In Evaluation of Transport Guarantees, Pre-dispatch Decision, Application Embedded Privilege Test, Jakarta Enterprise Beans Pre-dispatch Decision, and Jakarta Enterprise Beans Application Embedded Privilege Test, changed the description of how the checked permission is "obtained".
3. Added clarification of "the scope of a containers processing of a component request" to Policy Context Handlers.
4. Added a clarification to Policy Context Handlers, allowing containers to delay the registration of the required handlers.
5. In 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.
6. Added a footnote to Checking the Caller for a Permission, to clarify why calling Policy.getPermissions is not recommended.
7. Added Optimization of Permission Evaluations to describe the circumstances under which containers may caching the results of permission evaluations.

C.11.7. Changes to API

1. Added package description
2. Changed MethodSpec and constructor descriptions of EJBMethodPermission to provide support for additional method-intf values.
3. Clarified the syntax of typeName as used in methodParams of EJBMethodPermission. Also specified the corresponding affect on the canonical form returned by getActions.
4. 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.
5. 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.
6. In WebUserDataPermission, Fixed errors in the BNF for transportType.

7. 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.
8. 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

1. Updated the copyright dates.

C.11.9. Changes to Appendix B: Issues

1. Added descriptions of 3 new issues: TypeNames of EJBMethodPermission Array Parameters, Checking Permission on the root of a Web Application, and 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

1. Page 37: added requirement to Application Embedded Privilege Test to support calling isUserInRole from an unmapped (to servlet) web resource.
2. 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.
3. 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

1. Page 87: Clarified Description of WebRoleRefPermission class.
2. Page 88: Modified description of name parameter of WebRoleRefPermission constructor to describe use of empty-string name.

C.12.4. Changes to Appendix B: Issues

1. Page 105: removed sentence from description of resolution of issue B19, <<a830[See 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

1. Page 7: modified requirement 9 to allow for and describe the circumstances under which a container may run without a SecurityManager.
2. Page 8: added a154 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

1. 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.
2. Changed improper uses of "affect" to "effect".

C.14.2. Changes to Overview

1. In Assumptions, 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

1. 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

1. 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.
2. In What a Jakarta EE Platform's Deployment Tools Must Do 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.
3. In Servlet Policy Context Identifiers, described why each module of a multi-module web application must be deployed to a separate policy context.
4. In Translating Servlet security-role-ref Elements, clarified that the set of all roles defined for the

application is used to determine the additional permissions to be constructed.

5. In Jakarta Enterprise Beans Policy Context Identifiers, 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

1. Inserted new section Permission Names for Transport and Pre-Dispatch Decisions, 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.
2. Revised section Evaluation of Transport Guarantees and section Pre-dispatch Decision, to refer to the newly inserted section for the definition of their respective permission names.
3. Added new sentence the description of the EnterpriseBean Policy Context Handler 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

1. 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.
2. 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

1. Changed The specification version from 1.0 to 1.1

C.15.2. Changes to Policy Configuration Contract

1. Amended 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".
2. 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.

3. 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

1. Inserted new Matching HTTP Method Specifications to describe the HTTPMethodSpec as revised (by the definition of the HTTPMethodExceptionList) to support HTTP extension methods.
2. Modified WebResourcePermission Matching Rules and WebUserDataPermission Matching Rules to refer to the new section describing the matching of HTTP method specifications.

C.15.4. Changes to API

1. 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.
2. 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

1. Added new issue Support for HTTP Extension Methods. Resolution describes consequences with respect to backward compatibility:
2. Added new issue 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.
3. Added new issue 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 qualifying patterns.

C.16. Change log for Errata E

C.16.1. Changes Made Throughout the Document

1. Changed the specification version from 1.1 to 1.2

C.16.2. Changes to Overview

1. In Requirements, clarified requirement 4 to indicate that a policy provider in a Servlet or EJB only container need only satisfy the requirements corresponding to the supported container.
2. Corrected bullet 3 of a154, by removing prohibition on AccessControlContext.checkPermission.
3. Added new bullet 4 to a154, to ensure that container sets AccessControlContext if it uses the AccessController.checkPermission technique.

4. Added new section, Jakarta Servlet or Jakarta Enterprise Beans only containers, to differentiate requirements that must be satisfied by web containers from those that must be satisfied by EJB containers.

C.16.3. Changes to Policy Configuration Contract

1. in What a Jakarta EE Platform's Deployment Tools Must Do, modified the definition of equivalence to accept as equivalent a translation in which permissions that are implied by excluded permissions are removed from the role and unchecked permission collections. Limited the definition of equivalence to apply only to those permission types that are the subject of the translation. Added footnote to describe why equivalence cannot always be evaluated by PermissionCollection.implies().
2. in Translating Servlet Deployment Descriptors and in Translating Jakarta Enterprise Beans Deployment Descriptors, relaxed requirement that the value true be passed as the second argument to getPolicyConfiguration. Changed text to require that the policy statements be removed, and added footnotes to describe implementation choices.
3. added a requirement to "Qualified URL Pattern Names", that the translation use escaped encoding to differentiate colons occurring within the Pattern and QualifyingPattern elements from those used to construct the QualifyingPatternList.
4. Corrected determination of permission name in Translating Jakarta Enterprise Beans security-role-ref Elements such that the name is acquired from the ejb-name of the element containing the security-role-ref.
5. Added a new paragraph in Translating Jakarta Enterprise Beans security-role-ref Elements to describe the creation of additional EJBRoleRefPermission objects to support optional declaration of security-role-ref elements (as required by the EJB 3.0 specification)
6. Added a footnote to Translating Jakarta Enterprise Beans security-role-ref Elements to indicate that the requirements of this section apply to any elements that are permitted by the EJB deployment descriptor schema to contain security-role-ref elements. This was done in anticipation of support for inclusion of this element in the message-driven element

C.16.4. Changes to Policy Decision and Enforcement Contract

1. In Permission Names for Transport and Pre-Dispatch Decisions, added the requirement that all colon characters occurring within the name of the checked permission be represented using escaped encoding.
2. In Jakarta Enterprise Beans Pre-dispatch Decision corrected requirement that an RMISecurityException be thrown by requiring that the container throw an exception as required by the corresponding EJB Specification.
3. Added footnote to Policy Context Handlers to make it explicit that the requirement that a handler return a null value when called outside of the context of an invocation, need not apply to any additional handlers registered with the container.
4. Modified the requirements of Policy Context Handlers to allow containers to effectively delay registrations that would otherwise impede performance. As a result of the change, containers (especially EJB containers) may return null when, during the processing of a request, an attempt is made to invoke a required but not yet registered handler.

5. In Checking Grants corrected return result of AccessController.checkPermission when exception is not thrown.
6. Corrected the reference to the *javax.security.auth.Policy.getPolicy* method.

C.16.5. Changes to API

1. Added requirement to the HttpServletRequest based constructors of WebResourcePermission and WebUserDataPermission that the constructors must transform all colon characters occurring in the name to escaped encoding.
2. Added requirement that all colons occurring within the URLPattern elements of the name and URLPatternSpec arguments passed to the String based constructors of WebResoucePermission and WebUserDataPermission must be represented in escaped encoding.

C.16.6. Changes to Issues

1. Added recommended resolution to issue, Welcome File and security-constraint Processing.
2. Added resolution to issue, Colons Within path-segment of Request URI.

C.17. Change log for Errata F

C.17.1. Changes Made Throughout the Document

1. Changed the specification version from 1.2 to 1.3.

C.17.2. Changes to Policy Configuration Subcontract

1. in Translating security-constraint Elements, modified the translation to handle http-method-omission elements introduced by servlet 3.0.
2. added new "Combining HTTP Methods", to define the combination of http-method and http-method-omission elements, and to describe the translation of the results to the actions string used to construct WebResourcePermission and WebUserDataPermission objects.
3. In Example, modified the excluding auth constraint to demonstrate the use of an http-method-omission list. Also changed <<a416[See Permissions and PolicyConfiguration Operations from Example] to contain the corresponding translation.
4. In Translating Jakarta Enterprise Beans security-role-ref Elements, added an example in a footnote.

C.18. Change log for Errata G (maintenance Release 7)

C.18.1. Changes Made Throughout the Document

1. Changed the specification version from 1.3 to 1.4.
2. Changed the JCP version to 2.7

C.18.2. Changes to Policy Configuration Subcontract

1. In Policy Contexts and Policy Context Identifiers, added a footnote to describe exceptional case of EJBs bundled within a WAR.
2. In Servlet Policy Context Identifiers, added paragraph to ensure that EJBs defined in web modules are assigned to a separate policy context to ensure that the EJBcontext can be put in service before the policy context of the web module (which may depend on being able to call the EJB) . Also added a footnote with reference to section <<a512[See EJB Policy Context Identifiers] for further clarification.
3. Added Programmatic Servlet Registrations, to describe how the servlet policy translation defined by this subcontract can be applied to the security configuration resulting from the programmatic registration and security configuration enabled by Servlet 3.0. Also added a description of how an existing policy context may be retranslated while presrving its links to other policy contexts.
4. In Jakarta Enterprise Beans Policy Context Identifiers, added paragraph to ensure that EJBs defined in web modules are assigned to a separate policy context to ensure that the EJB context can be put in service before the policy context of the web module (which may depend on being able to call the EJB).
5. Clarified Deploying an Application or Module, to allow translations, links, and commits of individual modules to be interleaved as necessary to support runtime initialization of servlet policy (as required by Servlet 3.0) while preserving the ability of a ServletContextListener to make a local call to an EJB in the same application (and without getting an access exception).
6. In the optional Deploying to an existing Policy Configuration, added an additional paragraph to describe what must be done to capture the effects of any programmatic registrations and security configurations that may happen during initialization in a Servlet 3.0 container.
7. Simplified Redeploying a Module, by having it refer to Deploying an Application or Module, which, as described above, has been changed to handle Servlet 3.0.

C.18.3. Changes to API

1. Added clarification to removeUncheckedPolicy, removeExcludedPolicy, and removeRole methods (of the PolicyConfiguration interface) to indicate that these methods have no effect on the linkages among policy contexts.
2. Added requirement that the removeRole method of the PolicyConfiguration interface remove all roles when called with a role name of "*" and when no role by that name exists in the PolicyConfiguration..
3. Added clarification to the getPolicyConfiguration method of PolicyConfigurationFactory to indicate that it removes policy statements and linkages when the value of the remove parameter is true.

C.19. Change log for Errata H (maintenance Release 8)

C.19.1. Changes Made Throughout the Document

1. Changed the specification version from 1.4 to 1.5.
2. updated the license page

C.19.2. Changes to Policy Configuration Subcontract

1. In Programmatic Servlet Registrations, changed reference to Servlet 3.0 to "beginning with Servlet 3.0".
2. In Translating security-constraint Elements, added text to describe the handling of the role-name "**" in an auth-constraint, and to indicate that the "*" role does not imply the "**" role unless the application has defined its own role named "**". Amended description of permissions created for uncovered methods, to require that they be added to either the excluded or unchecked permission collections, based on the uncovered method semantic in effect for the web-module.
3. In Translating Servlet security-role-ref Elements, and Translating Jakarta Enterprise Beans security-role-ref Elements, added description of the handling of the "any authenticated user" role "**".
4. In Translating Jakarta Enterprise Beans method-permission Elements, amended description of the translation of role-names in method-permission elements, to include support for the role named "**".
5. In What the Provider Must Do, added requirement that the provider grant all permissions assigned to role "**" to any authenticated user.

C.20. Change log for Version 3.0

This is a major update of the specification, removing all references to the Java SE SecurityManager. Part of that removal is the dependency on the Java SE Policy, which has been replaced in this version by a similar Policy defined by Jakarta Authorization.