CHAPTER 2

Provider Configuration Subcontract

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

2.1 Policy Implementation Class

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

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

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

2.2 Permission Implementation Classes

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

2.3 Policy Configuration Interface

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

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

The abstract `PolicyConfigurationFactory` class and the `PolicyConfiguration` interface are defined in Chapter "API," which begins on page 53. Use of the `PolicyConfiguration` interface is defined in Chapter "Policy Configuration Subcontract," which begins on page 17.

2.4 PolicyContext Class and Context Handlers

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

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

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

The `PolicyContext` utility class and the `PolicyContextHandler` interface are defined in Chapter "API," which begins on page 53. Use of the `PolicyContext` class is defined in Chapter "Policy Configuration Subcontract," which begins on page 17.

2.5 What a Provider Must Do

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

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

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

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

2.6 Optional Provider Support for JAAS Policy Object

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

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

2.7 What the Application Server Must Do

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

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

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

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

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

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

```
            }
        }
```

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

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

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

2.7.1 Modifications to the JAAS SubjectDomainCombiner

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

To satisfy the contract defined by this specification, a J2EE 1.3 application server must install or bundle, such that it is used by every JRE of the application server, a `javax.security.auth.SubjectDomainCombiner` whose `combine` method returns protection domains constructed using the permission collections returned by `javax.security.auth.Policy.getPermisions`.

It is recommended that this requirement also be satisfied by J2EE 1.4 and later version Java EE application servers in the case where `javax.security.auth.Policy` is used (in backward compatibility mode) to perform javax.security.jacc policy decisions.