Before:

The following sample contains text I copied from the product Release Notes and edited lightly – primarily for consistency of style and active voice.

Security and Distributed Methods

Some distributed methods invoke other distributed methods on objects that are collocated in the same server. The container-generated remote implementations may perform these collocated invocations and custom methods can invoke collocated calls in entity or session beans.

All distributed methods are potentially subject to authorization checking. The actual authorization check occurs on the server side, in the container-generated implementations that delegate to the entity and session beans. This checking raises an important security issue:

*When using the deployment descriptor (**ejb-jar.xml**) to restrict access to specific distributed methods, the deployer must specify not only the distributed method that the client application explicitly invokes, but also ALL collocated distributed methods that are invoked by the explicitly invoked distributed method.*

An Example of Authorization Checking

The following situation illustrates this issue:

1. A client application invokes the standard container-generated **create**() method on an object reference for an entity bean's home interface.
2. The server does an authorization check and verifies the user/role/resource mapping for the client principal and the **create**() method resource.
3. The **create**() method internally invokes the container-generated **findByPrimaryKey**() and **findEJBHome**() methods.
4. The server does an authorization check for the client principal attempting to verify the user/role/resource mapping for the generated **findByPrimaryKey**() and **findEJBHome**() method resources as well.

For this situation, the deployer must specify (in the deployment descriptor) that the client principal has a role that has authorization to invoke **create()**, **findByPrimaryKey()**, and **findEJBHome()** even though the client principal does not explicitly invoke the latter two methods.

The ColocatedAuthorization Attribute

For custom methods in entity and session beans, this checking behavior is the correct behavior. However, for container-generated implementations, the deployer does not know what the container-generated methods call internally, so they should not have to specify allowing access to all the collocated distributed methods. To correct this difficulty, this version of PowerTier contains a **ColocatedAuthorization** attribute in the **.ptc** file.

The **ColocatedAuthorization** attribute has an **enabled** field that can be set to **true** or **false**. When the **enabled** field is **true**, all collocated distributed calls are subject to authorization checks. If the **enabled** field is **false**, only the top-level collocated distributed calls for container-generated methods are subject to authorization checks. The default value is **false**. The server **.ptc** file would look something like this:

```
<?xml version="1.0"?>
...
<ServerConfig>
    ...
    <JarFiles>
        ...
    </JarFiles>
    <SecurityPolicies>
        ...
        <ColocatedAuthorization
            enabled = "false"/>
    </SecurityPolicies>
    ...
</ServerConfig>
```

Authorization Checking and Container Methods

To continue the example from "An Example of Authorization Checking" on page 116, assume that **ColocatedAuthorization** has an **enabled** field of **false**.

1. A client application invokes the standard container-generated **create()** method on an object reference for an entity bean's home interface.
2. Since **create()** is a top-level container-generated distributed call, the server does an authorization check: verifying a user/role/resource mapping for the client principal and **create()** method resource.
3. The **create()** method internally invokes the **findByPrimaryKey()** and **findEJBHome()** methods.

You will notice in this scenario the server does no authorization checking on the **findByPrimaryKey()** and **findEJBHome()** methods. The server does no checking because these methods are container-generated methods and they are not top-level distributed invocations. In this situation, the deployer must specify (in thedeployment descriptor) that the client principal has a role that has authorization to invoke **create()** only.

Authorization Checking and Custom Methods

A slightly different situation occurs with custom entity or session bean methods that invoke collocated distributed methods. In this example, assume that **ColocatedAuthorization** has an **enabled** field of **false**:

1. The client application invokes the custom **createAccount()** method on an object reference for an entity bean's home interface.
2. Since **createAccount()** is a custom method, the server does an authorization check and verifies a user/role/resource mapping for the client principal and the **createAccount()** method resource.
3. The **createAccount()** method internally invokes a container-generated **create()** method that in turn internally invokes container-generated **findByPrimaryKey()** and **findEJBHome()** methods.
4. Since the **create()** method is the top-level container-generated method, the server does an authorization check: verifying a user/role/resource mapping for the client principal and **create()** method resource. However, the server does no authorization checking for **findByPrimaryKey()** and **findEJBHome()** since they are container-generated methods and are not top-level distributed invocations.
5. The **createAccount()** method then internally invokes another custom distributed method **matchIdentity()** on a collocated object.
6. Since **matchIdentity()** is a custom method and all custom methods are always subject to authorization checks, the server verifies a user/role/resource mapping for the client principal and **matchIdentity()** method resource.

In this situation, the deployer must specify (in the deployment descriptor) that the client principal has a role that has authorization to invoke the custom methods **createAccount()** and **matchIdentity()**, as well as the top-level container-generated method **create()**.

After:

At the request of a reviewer, I modified the textual examples and created graphical representations.

Security and Distributed Methods

Some distributed methods invoke other distributed methods on objects that are collocated in the same server. Some of these calls may be made automatically by methods generated by PowerTier. In addition, custom methods can make collocated calls in entity or session beans.

All distributed methods are potentially subject to authorization checking. The server performs the actual authorization check, in the generated methods that delegate to the entity and session beans. This checking raises an important security issue:

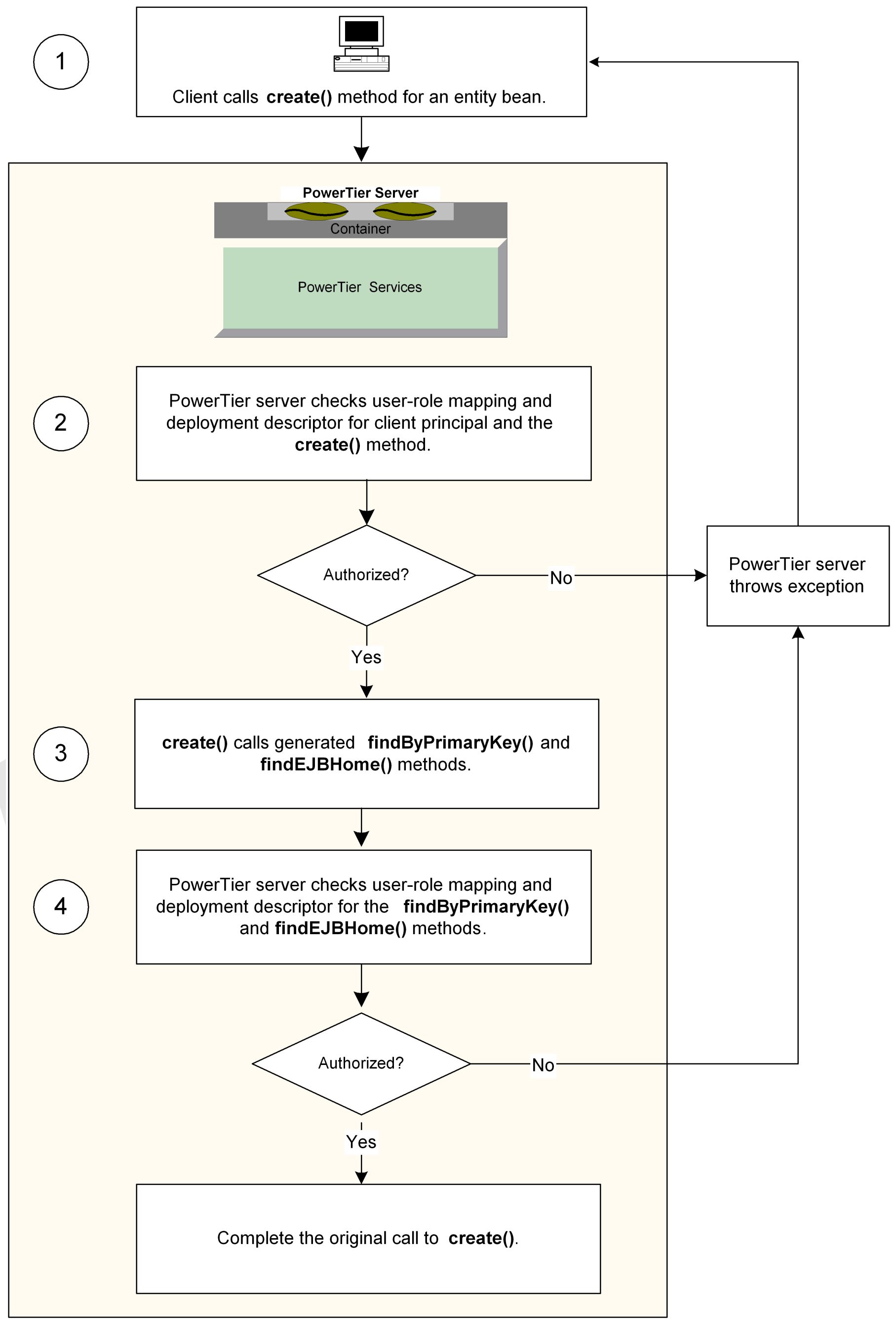
> When using the deployment descriptor (**ejb-jar.xml**) to restrict access to specific distributed methods, the deployer must specify not only the distributed method that the client application explicitly invokes, but also *all* collocated distributed methods that the distributed method itself invokes explicitly.

For information about PowerTier-generated methods, custom business logic, and client-side methods, see *Planning Your Enterprise Architecture* and the *PowerTier Server and EJB Development Guide*.

Figure 7 shows an example that illustrates this situation.

In this example, the deployer must specify (in the deployment descriptor) that the client principal has a role that has authorization to invoke the **create**(), **findByPrimaryKey**(), and **findEJBHome()** methods, even though the client principal only explicitly invokes the **create**() method.

Figure 7. Authorization Checking for Generated Distributed Methods


1
Client calls create() method for an entity bean.
PowerTier Server
Container
PowerTier Services
2
PowerTier server checks user-role mapping and deployment descriptor for client principal and the create() method.
Authorized?
No
PowerTier server throws exception
Yes
3
create() calls generated findByPrimaryKey() and findEJBHome() methods.
4
PowerTier server checks user-role mapping and deployment descriptor for the findByPrimaryKey() and findEJBHome() methods.
Authorized?
No
Yes
Complete the original call to create().

The ColocatedAuthorization Element

For custom methods in entity and session beans, this checking behavior is the correct behavior. For generated methods, however, the deployer may not know which other methods the generated methods call. In this case, it should not be necessary to explicitly allow access to all of the collocated distributed methods. Therefore, the PowerTier server configuration (**.ptc**) file contains a **ColocatedAuthorization** element, which prevents this problem.

The **ColocatedAuthorization** element has an attribute called **enabled** that can be set to either **true** or **false**:

- When **enabled** is set to **true**, all collocated distributed calls are subject to authorization checks.
- When **enabled** is set to **false**, only the top-level collocated distributed calls for generated methods are subject to authorization checks.

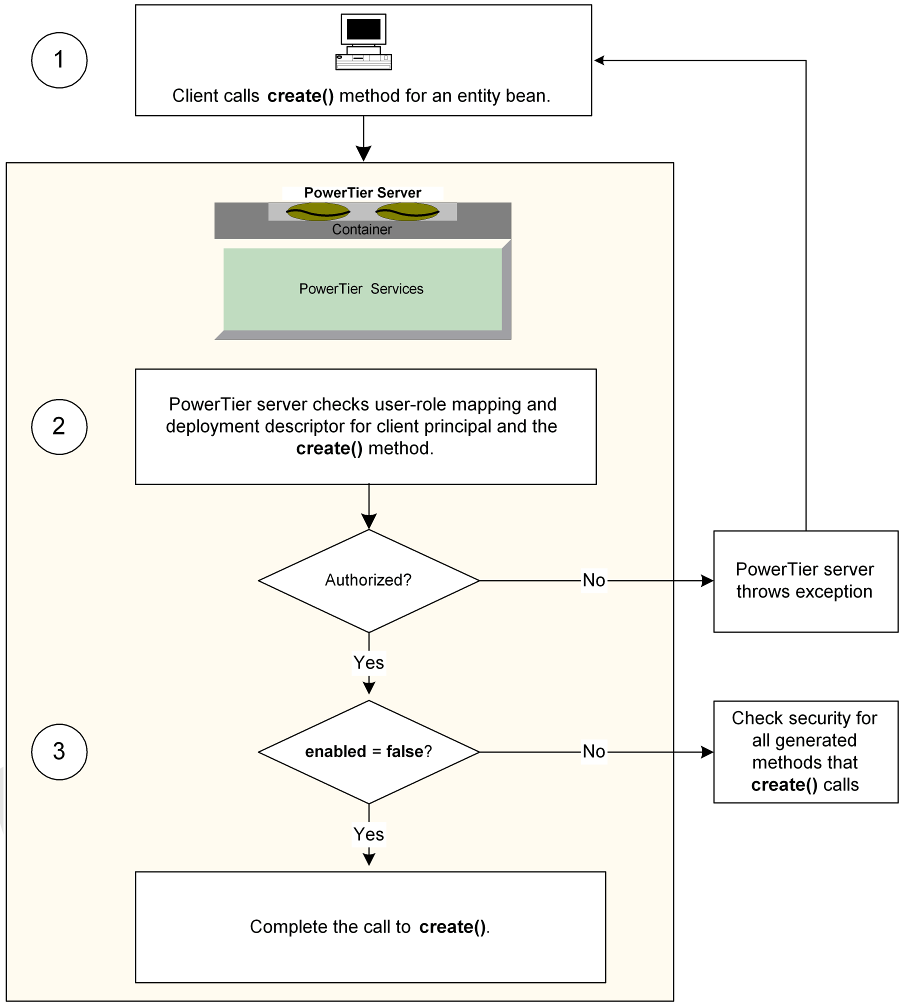
The default value is **false**. The **.ptc** file would look something like this:

```
<?xml version="1.0"?>
...
<ServerConfig>
    ...
    <SecurityPolicies>
        ...
        <ColocatedAuthorization
            enabled = "false"/>
    </SecurityPolicies>
    ...
</ServerConfig>
```

Figure 8 extends the example in Figure 7 in the case where the **enabled** attribute of the **ColocatedAuthorization** element is set to **false**.

In this case, the server does no checking of either **findByPrimaryKey()** or **findEJBHome()** because these are generated methods. The deployer only needs to specify that the client principal has a role that has authorization to invoke **create()**.

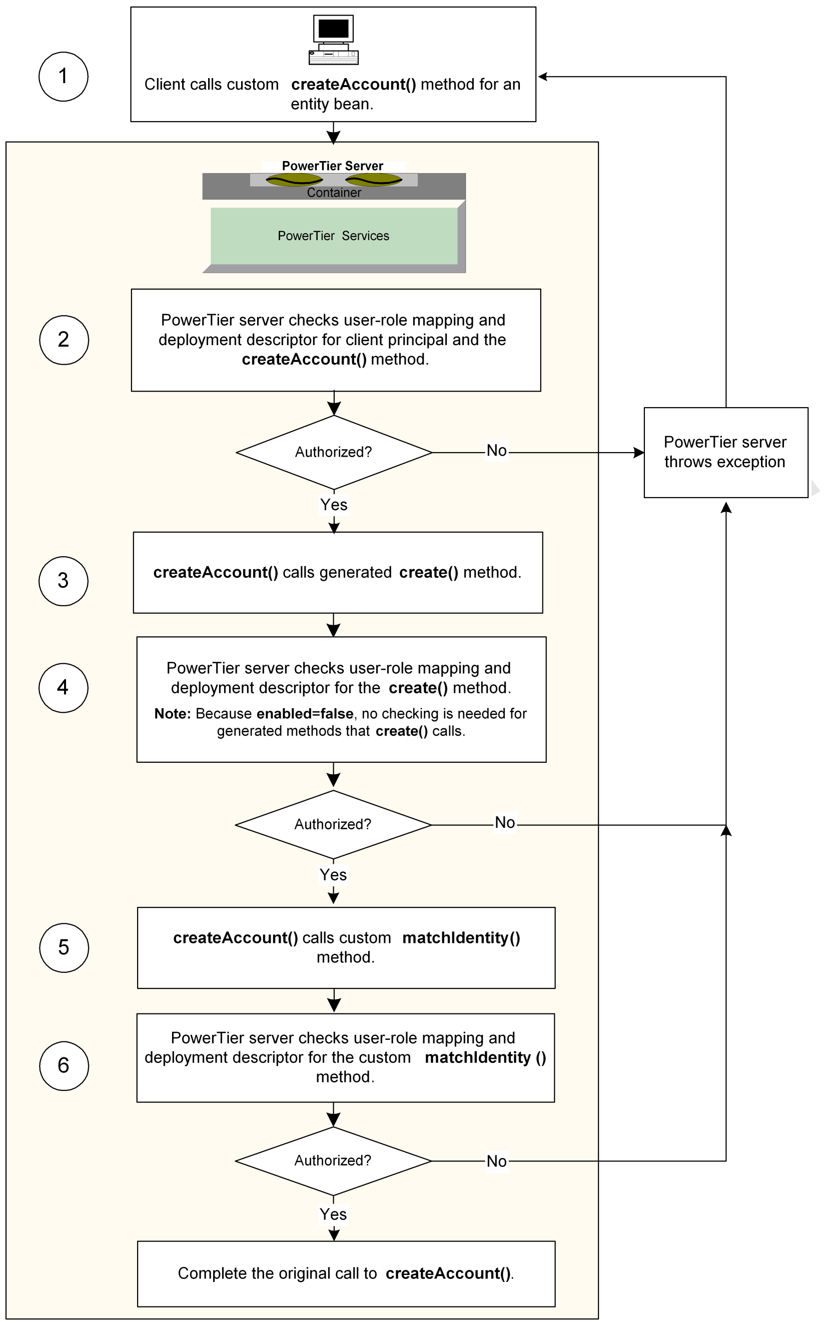
Figure 8. Authorization Checking With ColocatedAuthorization, enabled=false


1
Client calls create() method for an entity bean.
PowerTier Server
Container
PowerTier Services
2
PowerTier server checks user-role mapping and deployment descriptor for client principal and the create() method.
Authorized?
No
PowerTier server throws exception
Yes
3
enabled = false?
No
Check security for all generated methods that create() calls
Yes
Complete the call to create().

Authorization Checking and Custom Methods

A slightly different situation occurs with custom entity or session bean methods that invoke collocated distributed methods. Figure 9 shows an example where the custom method **createAccount**() calls both the generated **create**() method and the custom method **matchIdentity**(). In this example, the **enabled** attribute of the **ColocatedAuthorization** element is set to **false**.

Figure 9. Authorization Checking for Custom Distributed Methods


1
Client calls custom **createAccount()** method for an entity bean.
PowerTier Server
Container
PowerTier Services
2
PowerTier server checks user-role mapping and deployment descriptor for client principal and the **createAccount()** method.
Authorized?
No
PowerTier server throws exception
Yes
3
createAccount() calls generated **create()** method.
4
PowerTier server checks user-role mapping and deployment descriptor for the **create()** method.
Note: Because **enabled=false**, no checking is needed for generated methods that **create()** calls.
Authorized?
No
Yes
5
createAccount() calls custom **matchIdentity()** method.
6
PowerTier server checks user-role mapping and deployment descriptor for the custom **matchIdentity ()** method.
Authorized?
No
Yes
Complete the original call to **createAccount()**.