Authorization in Plugins

In This Topic

Default Authorization

By default a plugin does not define any authorization which means that all components contained by the plugin can be used by anyone at any time (unless a component is licensed and the application server on which it is installed does not have the proper license). This is normally done during development and is not recommended in a live situation.

Back to Top

Permission Category

In XperienCentral, a plugin may define zero or more permissions. A permission is a fine-grained definition of a particular operation on one or more objects. Assigning the permission to a role grants that role the particular permission. A permission must always be positive - it grants particular rights and never denies rights.

Permissions always belong to one particular permission category. A plugin may define zero or more permission categories, at most one for each component it contains. However, different components may use one and the same permission category.

The first step in creating custom authorization is to define the permission categories. A permission category has the following properties:

Property	Purpose
`Value`	A unique identifier for the permission category.
`labelId`	ID of the label in `messages_<language>_<country>.properties` containing the translation of the name of the permission category in the supported language(s).
`showAsComponent`	Indicates whether this permission category appears in the channel configuration panel of the Workspace. If it is visible, the plugin as a whole can be enabled/disabled per channel.

After creating an instance of the permission category and setting its properties, use `setPermissionCategory` on the component definition to assign the category to a component definition. If you reuse the same permission category for another component, you should set the permission category only once. If you define the permission category twice, this will result in a conflict. For example, to define a permission category for a “Plugin Management console” panel, you would use the following:

```
PermissionCategoryImpl
wcbManConsoleCategory = new PermissionCategoryImpl();
wcbManConsoleCategory.setValue(WCB_MANAGMENT_CONSOLE_COMPONENT_ID);
wcbManConsoleCategory.setLabelId(WCB_MANAGMENT_CONSOLE);
wcbManConsoleCategory.setShowAsComponent(true);
panelDefinition.setPermissionCategory(wcbManConsoleCategory);
```

When the plugin is deployed, the permission categories defined in each component definition contained by the plugin are installed or updated automatically. When defining and using permission categories, be sure that they conform to the development guidelines (G050 and G051 in particular).

Back to Top

Permissions

A permission grants a role the rights to perform a particular operation on one or more objects. This is a definition that can be interpreted in many ways, and the permission itself does not define the exact operation it grants permission for and the object on which the operation operates. It is the software itself using the permission that defines what rights the permission provides.

A permission consists of the following properties:

Property	Purpose
`value`	Unique identifier for the permission.
`labelID`	ID of the label in `messages_<language>_<country>.properties` containing the translation of the name of the permission in the supported language(s).
`permissionCategory`	The permission category to which the permission belongs.

To instantiate permissions invoke the constructor of `PermissionImpl`. The `value`, `labelId` and `permissionCategory` are input arguments of the constructor. The permissions defined in each permission category of the plugin are installed or updated automatically when the plugin is deployed. When defining and using permissions, be sure that they conform to the development guidelines (G052 and G053 in particular).

Back to Top

Permission Groups

Because of the fine-grained definition of permissions, assigning them to the proper roles can sometimes be difficult. The default XperienCentral application itself already contains over 70 separate permissions and this amount will only grow in the future or when you deploy additional plugins. For ease of use, the concept of a permission group was introduced. A permission group is nothing more than a collection of permissions. Instead of assigning individual permissions to a role, it is also possible to assign a permission group to a role, thus implicitly assigning a collection of permissions to the role.

Permissions defined in a plugin can be added to such a permission group using the method `setPermissionsForPermissionGroup` on the component definition. XperienCentral defines five different permission groups:

Permission group ID	Purpose
`CASUAL_USER_PERMISSION_GROUP`	Permission group for casual users
`EDITOR_PERMISSION_GROUP`	Permission group for editors
`MAIN_EDITOR_PERMISSION_GROUP`	Permission group for main editors
`APPLICATION_MANAGER_PERMISSION_GROUP`	Permission group for Application managers
`DEVELOPER_EDITOR_PERMISSION_GROUP`	Permission group for developers

The class `nl.gx.webmanager.wcb.WCBConstants` contains the identifiers for these groups. The code example below shows how to add a collection of permissions to a permission group:

```
PermissionImpl[] allPermissions =  new PermissionImpl[] { openPanelPermission, startStopPermission,
installUpdatePermission};
panelDefinition.setPermissionsForPermissionGroup(WCBConstants.CASUAL_USER_PERMISSION_GROUP, noPermissions);
```

When the plugin is deployed, the permissions are automatically added to the permission group and thus implicitly to all roles assigned to this permission group. This way even a plugin that defines restricted access can become available to users automatically without manual intervention.

Back to Top

Using Permissions

To use permissions in your code you first have to define a dependency with the Authorization service. The Authorization service provides a method `checkAccess` which takes the permission's value as an input argument in order to check whether the current user has the specific permission. Depending on the result of this call a particular piece of software will or will not be invoked.

The code snippets below show how to define the dependency with the authorization service in the activator followed by an authorization check.

```
ComponentDependencyImpl
authDependency = new ComponentDependencyImpl();
authDependency.setServiceName(AuthorizationService.class.getName());
authDependency.setRequired(true);
panelDefinition.setDependencies(new ComponentDependency[]{authDependency};
```

```
if (authorizationService.checkAccess(CREATE_BOOK)) {
        return createBook();
}
else {
        LOG.warning("User does not have authorization to create books");
        return null;
}
```


CREATE_BOOK in the code snippet above refers to a `public static final String` defined once. It is a good code practice to define such static String definitions in this manner.

To check authorization from a JSP use the `wmedit:checkPermission` tag. Input arguments are the permission value and ID of the website. The code snippet below shows an example how to conditionally print the text "user is authorized to generate PDF" for the permission with value `pdf_generate`:

```
<c:set var="website" value="${editcontext.website}" />
<wmedit:checkPermission value="pdf_generate" website="${website.id}">
User is authorized to generate PDF </wmedit:checkPermission>
```

When using permissions, be sure that they conform to the development guidelines (G052 and G053 in particular).

Back to Top

Element Permission

For element components, the element itself is created and deleted by the XperienCentral framework rather than the controller contained by the element component itself. While create and delete authorization checks would typically be programmed in a Java class of the plugin, for element components this is done differently.

To define the permissions that grant the rights to create an instance of the element, invoke `setCreatePermissions` on the element component definition. To define the permissions that grant the rights to delete an instance of the element, invoke `setDeletePermission`.

For example for a BookElement it would look like this:

```
Permission[]
createPermissions = new Permission[]{createPermission};
elemCompDef.setCreatePermissions(BookElement.class, createPermissions);

Permission[]
deletePermissions = new Permission[]{deletePermission};
elemCompDef.setDeletePermissions(BookElement.class, deletePermissions);
```

When defining and using element permissions, be sure that they conform to the Plugin Development Guidelines (G052 and G053 in particular).