Plugin Development: Permissions

You may wish to restrict some actions or views in your plugin to certain operator roles only, and this can be achieved by using the permissions system.

Each permission has four possible attributes that can be set.

Attribute	Description
view	Used to allow the operator to access a GET route.
create	Used to allow the operator to create a new resource, typ
update	Used to allow the operator to update an existing resourc
delete	Used to allow the operator to delete a resource, typically

To add a single permission for the plugin, use the addPermission function. The name must be alphanumeric. The attributes is an array keyed by the above elements and with a boolean value, and the third param is the language key to show in the Roles section. For example, you should create a language string such as 'Manage Settings' and use the equivalent language key.

```
$this->addPermission($name, $attributes, $lang);
```

For example, if we want to add a permission to allow managing the settings.

```
$attributes = [ 'view' => true, 'create' => true, 'update' => true,
```

```
'delete' => true ];$this->addPermission('settings', $attributes,
'Plugins#HelloWorld::lang.permission');
```

A fourth parameter can be entered for the array of role IDs that should have this permission initially. If not entered, it will automatically include the System Administrators role.

```
$attributes = [ 'view' => true, 'create' => true, 'update' => true,
'delete' => true ];$this->addPermission('settings', $attributes,
'Plugins#HelloWorld::lang.permission', [1, 2, 3]);
```

The examples above would create a permission called 'helloworld_settings', as the plugin name is automatically preprended to the permission name you set.

Removing a Permission

To remove a single permission with a known name for the plugin, use the removePermission function.

```
$this->removePermission($name);
```

To remove all permissions for the plugin, for example when deactivating/uninstalling the plugin, use the removePermissions function.

```
$this->removePermissions();
```

Checking if a Permission Exists

If you need to just check if a permission with a known name currently exists, you can use the hasPermission function, which returns a Boolean.

```
$this->hasPermission($name);
```

To check if the plugin has any permissions registered currently, you can use the hasPermissions function, which returns a Boolean.

```
$this->hasPermissions();
```

Utilizing Use Permission
Protecting Routes

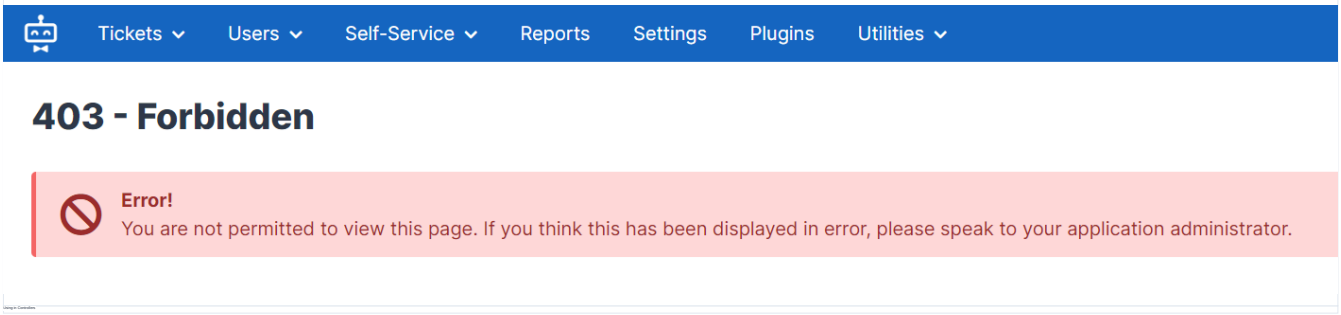
The easiest way of using permissions is to protect specific routes, this can be done with the can key in the routes attributes. The value should be the permission attribute to check (from the table above), followed by the name of the permission. The below example uses a single permission but separate attributes on the GET and POST call for the settings page.

Routes/plugin.php

```
<?phpRoute::get('settings', [    'as'   =>
 'plugin.helloworld.settings',    'can'  =>
 'view.helloworld_settings',    'uses' =>
 'Addons
PluginsHelloWorl
dControllersHelloWorld@getSetting
```

```
sPage']);Route::post('settings', [    'as'   =>
 'plugin.helloworld.settings.update',    'can'  =>
 'update.helloworld_settings',    'uses' =>
 'AddonsPluginsHelloWorldControllersHelloWorld@updateSettings']);
```

If an operator doesn't have the specified permission, they will see a 403 forbidden page when trying to access the route.



To check if an operator has a specified permission in your controller code, you can use the hasPermission method on the User model.

```
auth_user()->hasPermission(
'view.helloworld_settings'); // Will return true/false
```

Similar to above, you can use the hasPermission method on the User model in your views too.

```
{% if auth_user().hasPermission(
'view.helloworld_settings') %}    Permission exists{% endif %}
```

Online URL:
https://docs.supportpro.vn/article/plugin-development-permissions-215.html