Common-Controls Security

Version 1.0.2 - Stand: 24. December 2003

Published by:
SCC Informationssysteme GmbH
64367 Mühltal

Tel: +49 (0) 6151 / 13 6 31 0
Internet www.scc-gmbh.com

Product Site
http://www.common-controls.com

Table of content

1 Introduction **1**

2 Authorization types **2**

3 Implementation of an Authorization System **3**
3.1 Providing the Principal Object 3
3.2 Linking to an Authorization System 4
3.3 Registration of the Principal Object 5
3.4 Configuration of Authorizations 6
3.5 Securing the Action classes in the struts-config.xml 7

4 Examples **9**

5 Support **10**

6 Glossar **11**

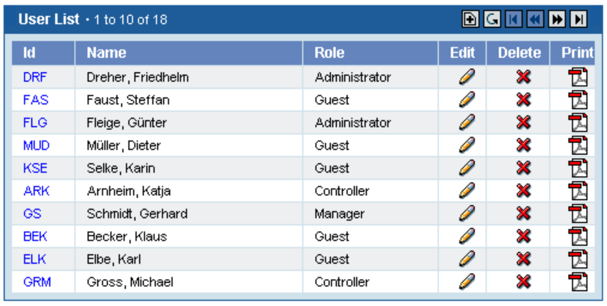
1 Introduction

This document describes how authorizations can be granted at the level of control elements, in order to restrict program functions to certain groups of users only. For this purpose, a control element has a **permission**-attribute, which allows assignment of the necessary authorizations (see Figure 1). The authorizations can then be role-based or function-based. The various authorization types are shown in Section 2.

The following example shows a function-based authorization system. If the user does not have the rights to create (create), edit (edit), delete (delete) or print (print) a data record, he will not be displayed the corresponding buttons and columns (see Figure 2).

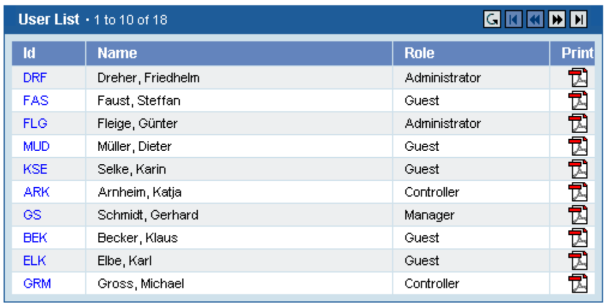
```
<%@ taglib uri="/WEB-INF/tlds/cc-controls.tld" prefix="ctrl" %>

<ctrl:list
     id="list1"
     action="sample101/userBrowse"
     name="users"
     title="User List"
     width="500"
     rows="10"
     refreshButton="true"
     createButton="$create">

     <ctrl:columndrilldown   title="Id"       property="userId"       width="65"/>
     <ctrl:columntext        title="Name"     property="name"         width="350"/>
     <ctrl:columntext        title="Role"     property="role.name"    width="150"/>
     <ctrl:columnedit        title="Edit"     permission="$edit"/>
     <ctrl:columndelete      title="Delete"   permission="$delete"/>

     <ctrl:columnbutton      title="Print"    property="print"
         permission="$print"
         image="app/images/imgPDF.gif"
         align="center"
         target="_blank"/>
</ctrl:list>
```

Figure 1: Configuration of rights for a control element



User List · 1 to 10 of 18

Id	Name	Role	Edit	Delete	Print
DRF	Dreher, Friedhelm	Administrator	✎	✖	📄
FAS	Faust, Steffan	Guest	✎	✖	📄
FLG	Fleige, Günter	Administrator	✎	✖	📄
MUD	Müller, Dieter	Guest	✎	✖	📄
KSE	Selke, Karin	Guest	✎	✖	📄
ARK	Arnheim, Katja	Controller	✎	✖	📄
GS	Schmidt, Gerhard	Manager	✎	✖	📄
BEK	Becker, Klaus	Guest	✎	✖	📄
ELK	Elbe, Karl	Guest	✎	✖	📄
GRM	Gross, Michael	Controller	✎	✖	📄



User List · 1 to 10 of 18

Id	Name	Role	Print
DRF	Dreher, Friedhelm	Administrator	📄
FAS	Faust, Steffan	Guest	📄
FLG	Fleige, Günter	Administrator	📄
MUD	Müller, Dieter	Guest	📄
KSE	Selke, Karin	Guest	📄
ARK	Arnheim, Katja	Controller	📄
GS	Schmidt, Gerhard	Manager	📄
BEK	Becker, Klaus	Guest	📄
ELK	Elbe, Karl	Guest	📄
GRM	Gross, Michael	Controller	📄

Figure 2: Display of ListControl with different authorizations

The figure on the left shows the list the way it is displayed to a user with the function rights to ***edit***, ***delete***, ***print*** and ***create***. The user on the right-hand page only has the function authorization ***print***.

2 Authorization types

The Framework distinguishes between the following authorizations:

- StaticPermission
- RoleBasedPermission
- FunctionBasedPermission
- Own implementations

2.1.1 StaticPermissions

A static authorization defines whether the user may carry out an action or not. In consequence, only the values “true“ or “false” can be assigned as the variants.

Example: refreshButton=“true“

2.1.2 RoleBasedPermission

In the case of a role-based authorization, the execution of an action is linked to a role of the user. Thus, in this manner, access to master data maintenance can be generally restricted to administrators.

A role-based authorization is marked by means of the “**#”** character.

Examples: **#admin; #gast; #manager**.

2.1.3 FunctionBasedPermission

In the case of a function-based authorization, the execution of an action is linked to a function. The user, therefore, must have the right to carry out a special function. In the simplest case, this includes e.g. the deletion, viewing, printing or addition of a data record.

Functions can thus be defined in a general form or also for individual sections within an application. Thus, e.g., it is conceivable that a user may be able to insert data records in the persons table (User), but that he does not have this right in the clients table (Client).

A function-based authorization is identified by the “**$**” character.

Examples: **$user.edit; $user.create; $user.delete; $client.edit;**

2.1.4 Own Implementations

Apart from the standard types, you can also realize your own authorization types.

2.1.5 Access Control List

Authorizations are always specified in the form of an Access Control List (**ACL**). What is involved here is a semicolon-delimited list with individual authorizations. Various different authorization types can be used in a list.

Example: **$user.edit;#admin**

If this list is then stored with authorizations within a ListControl on the edit-column, then, e.g. the column can only be displayed if the user concerned is an Administrator or he has access to the function “user.edit”.

The assignment of rights to a user is then done in a database or a separate configuration file.

3 Implementation of an Authorization System

The following steps are required for using an authorization system in conjunction with the Common Controls Framework:

1. Providing the principal object
2. Registration of the principal object
3. Linking to an authorization system
4. Configuration of authorizations within the control elements
5. Securing the action classes in the struts-config.xml

3.1 *Providing the Principal Object*

If an authorization check is to be carried out within the control elements, the application must make available a **`Principal Object`** for the purpose. Only the `Principal` Interface is implemented for this purpose. It extends an existing class with methods that are required for checking authorizations. This could involve, as shown in the following examples, the user object, which is generated for a user at the time of system login (in this context, under 3.3).

```java
import com.cc.framework.security.Principal;
import com.cc.sample.config.SecurityConfig;
import com.cc.sample.security.UserRole;

/**
 * User-Objekt
 */
public class User implements Principal {

	private String userId = "";

	/**
	 * Constructor
	 * @param  userId     The UserId
	 */
	public User(String userId) {
		super();

		this.userId = userId;
	}

	/**
	 * @see com.cc.framework.security.Principal#hasRight(java.lang.String)
	 */
	public boolean hasRight(String right) {
		// This methode is called if a function based permission
		// is checked
		// If the user can perform the action return true, false otherwise.
	}

	/**
	 * @see com.cc.framework.security.Principal#isInRole(java.lang.String)
	 */
	public boolean isInRole(String role) {
		// This methode is called if a role based permission
		// is checked
		// If the user is in role return true, false otherwise.
	}
}
```

CodeSnippet 1: User Objekt

3.2 Linking to an Authorization System

The linking to an authorization system takes place in the methods Principal#hasRight(String right) and Principal#isInRol(String role) and is the job of the application. Here, e.g. access may be made to an LDAP Server or a user database.

For example, the Online Demo 2 here uses a simple authorization system which can be configured by means of XML-files and is sufficient for small applications.

3.3 Registration of the Principal Object

For the administration of the Principal Object, the class **SecurityUtil** provides the following methods:

Method	Description
getPrincipal(...)	Returns the principle object stored in the session.
registerPrincipal(...)	Registers the principal object in the session under the key Globals.PRINCIPAL_KEY.
unregisterPrincipal(...)	Deletes the registered principal object from the session.

Tabelle 1: The class SecurityUtil

The following example code shows the registration of a user object as the principle object after the authentication of the user.

```
import com.cc.framework.security.SecurityUtil;
import com.cc.framework.adapter.struts. FWAction ;
import com.cc.framework.adapter.struts.ActionContext;
import com.cc.framework.adapter.struts. FormActionContext;

public class LogonAction extends FWAction {


     public void doExecute(ActionContext ctx) throws IOException, ServletException {
         ctx.forwardToInput();
     }

     public void logon_onClick(FormActionContext ctx) {

         User user = new User(form.getUserId());
         user.load();

         // check if the user is authorized for this application

         // 2) register the user object as the principal object
         SecurityUtil.registerPrincipal(ctx.session(), user);

     }
}
```

CodeSnippet 2: Registration of the Principal Object

A registered principle object is de-registered by calling the static method `unregisterPrincipal()` (e.g. when the user logs out).

```
SecurityUtil.unregisterPrincipal(ctx.session());
```

3.4 *Configuration of Authorizations*

3.4.1 Control Elements

For control elements, authorizations can be assigned at the following levels:

Control element	
ListControl	- Displaying or suppressing the control element itself - Add-Button; displaying or suppressing - Column, displaying or suppressing (for all types of columns)
tree	- Displaying or suppressing the control element itself No special mechanism is currently being offered at the level of the nodes. The application itself must therefore ensure that the data model only contains nodes that may be displayed to the user.
TreeListControl	- Displaying or suppressing the control element itself - Add-Button; displaying or suppressing - Column, displaying or suppressing (for all types of columns) No special mechanism is currently being offered at the level of the nodes. The application itself must therefore ensure that the data model only contains nodes that may be displayed to the user.
Tabset	- Displaying or suppressing the control element itself A tab of the tabset is disabled using the “enabled” attribute. If certain tabs are not to be displayed, then at present, this must be ensured by the application itself.
MenuControl	- Displaying or suppressing the control element itself - Menu item, displaying or suppressing

3.4.2 Tags

Moreover, the Common Controls TagLibrary provides two additional tags, using which the execution of a section in a JSP page can be controlled in an authorization-dependent manner.

- <sec:granted>
- <sec:notGranted>

The `<granted>` Tag lists the contents of the Tag body only if the registered principal object has the required authorization.

```
<%@ taglib uri="/WEB-INF/tlds/cc-security.tld" prefix="sec" %>

<sec:granted permission="#admin;#developer">
       This User has the admin- or developer Role.
</sec:granted>
```

The `<notGranted>` Tag lists the contents of the Tag-body only when the registered principal object **does not** have the required authorization.

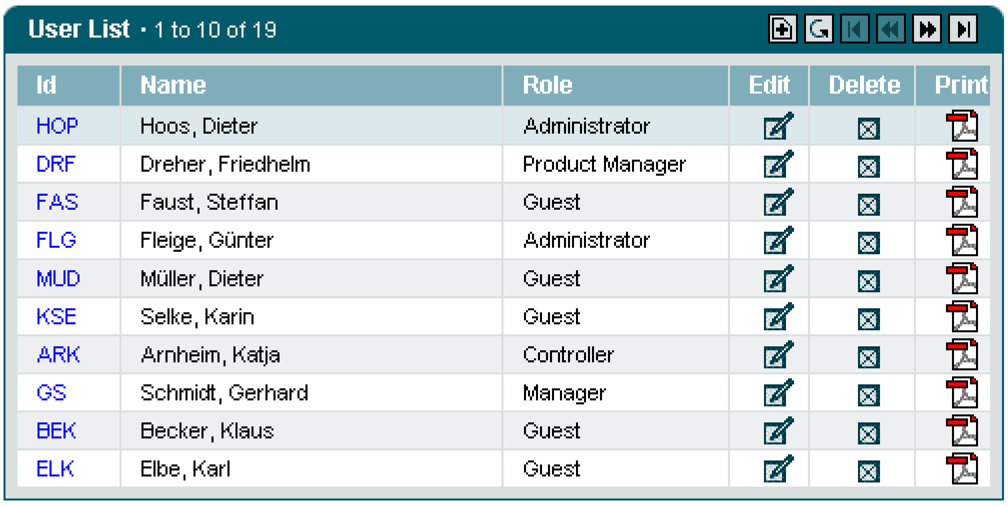
```
<%@ taglib uri="/WEB-INF/tlds/cc-security.tld" prefix="sec" %>

<sec:notGranted permission="#admin;#developer">
       This User has not the admin- or developer Role.
</sec:notGranted>
```

Here, the function-based and role-based rights can also be mixed at will.

3.5 Securing the Action classes in the struts-config.xml

By default, the control elements generate hyperlinks which result in the execution of an action on the server. The parameters required for this purpose are transferred in the URL within the request.

The following example shows this for the “Delete” column of a list.



User List · 1 to 10 of 19

Id	Name	Role	Edit	Delete	Print
HOP	Hoos, Dieter	Administrator			
DRF	Dreher, Friedhelm	Product Manager			
FAS	Faust, Steffan	Guest			
FLG	Fleige, Günter	Administrator			
MUD	Müller, Dieter	Guest			
KSE	Selke, Karin	Guest			
ARK	Arnheim, Katja	Controller			
GS	Schmidt, Gerhard	Manager			
BEK	Becker, Klaus	Guest			
ELK	Elbe, Karl	Guest			

Generated URL when deleting the data record with the Id = HOP:

http://localhost:8080/cc/listcontrol/sample101/userBrowse.do?ctrl=user&action=delete&param=HOP

Such URLs can also be entered by unauthorized users in the address bar of the browser, and executed, regardless of whether the user has an enabled Delete button or not. Therefore, the action that is triggered with the hyperlink must itself be secured once again on the server side. Here, in the `struts-config.xml` file, the **`roles`**-attribute is used for this purpose, to which a role-based or function-based right can be assigned. The following extract from a `struts-config.xml` file shows how the authorization check is expanded to cover the execution of an action class.

Example: Configuration of Authorizations

```
<struts-config>

    <action
        path="/sample101/userEdit"
        name="userEditForm"
        scope="request"
        validate="false"
        roles="$user.edit"
        type="com.cc.sample.list.sample101.action.UserEditAction"
        input="/../jsp/list/sample101/UserEdit.jsp">
        <forward name="back"     path="/sample101/userBrowse.do" redirect="true"/>
        <forward name="success"  path="/sample101/userBrowse.do" redirect="true"/>
    </action>

    <action
        path="/sample101/userCreate"
        name="userCreateForm"
        scope="request"
        validate="false"
        roles="$user.create"
        type="com.cc.sample.list.sample101.action.UserCreateAction"
        input="/../jsp/list/sample101/UserCreate.jsp">

        <forward name="back"    path="/sample101/userBrowse.do" redirect="true"/>
        <forward name="success" path="/sample101/userBrowse.do" redirect="true"/>
    </action>

    <action
        path="/sample101/userDelete"
        roles="$user.delete"
        type="com.cc.sample.list.sample101.action.UserDeleteAction">

        <forward name="back" path="/sample101/userBrowse.do" redirect="true"/>
    </action>

    <action
        path="/sample101/userPrint"
        roles="$user.print"
        type="com.cc.sample.list.sample101.action.UserPrintAction">

        <forward name="back" path="/sample101/userBrowse.do" redirect="true"/>
    </action>

</struts-config>
```

In order that the configured rights can be checked against the registered principle object, in the `struts-config` file, the following class must additionally be entered as a RequestProcessor

```
<struts-config>

      ...

      <!-- RequestProcessor -->
      <controller
            nocache="true"
            processorClass="com.cc.framework.adapter.struts.FWRequestProcessor"/>

      ...

</struts-config>
```

The **FWRequestProcessor** class is derived from the Struts RequestProcessor class and overwrites the `processRoles`() method of the higher class, to facilitate the checking of the `roles` attributes with the help of the registered principle object.

4 Examples

A detailed example of the configuration of authorizations and the use of the principal object is given in the examples for the Online Demo 2.

5 Support

We would be happy to be of service if you have any questions or problems. Please use our Service Form on our homepage for your queries. We shall endeavor to answer your queries as quickly as possible.

6 Glossar

A

ACL	Access Control List

C

CC	Common-Controls