COMMON CONTROLS

Security

Restricting program functions to certain groups of users

SCC Informationssysteme GmbH

Version 1.6 · As of: 14. Januar 2006

Impressum

PUBLISHED BY

SCC Informationssysteme GmbH
Drosselweg 13
64367 Mühltal
+49 6151 13 63 10
www.scc-gmbh.com

Contents

1 Introduction

This document describes how permissions can be assigned at the level of individual control elements in order to restrict program functions to certain groups of users. For this purpose, every control element has a permission attribute to which the required permissions are assigned (see figure 1). These may be role-based or function-based permissions. The different types of permission are presented in section 2.

The following example shows a function-based permission system. If the user does not hold the rights to create, edit, delete or print a record, the corresponding buttons and columns are not displayed (see figure 2).

JSP

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

<ctrl:list
     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>
```

Configuring permissions for a control element

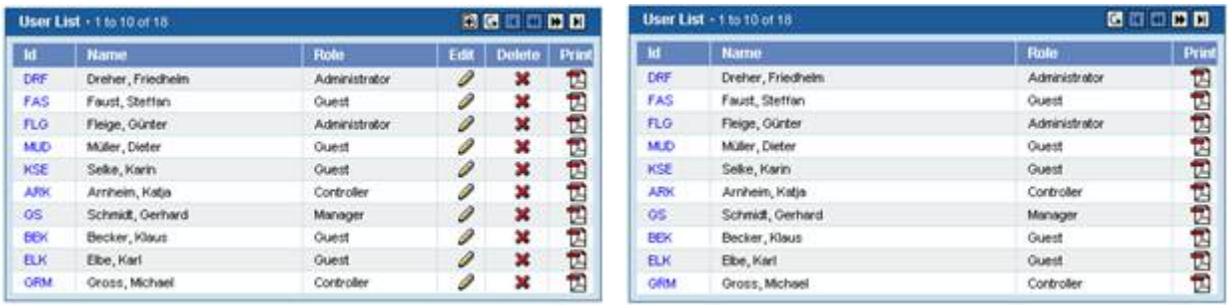

Figure 1: A ListControl as seen with different permissions

The picture on the left shows the list as it appears to a user holding the function permissions edit, delete, print and create.

The user on the right holds the print permission only.

2 Permission types

The framework distinguishes the following kinds of permission:

- StaticPermission
- RoleBasedPermission
- FunctionBasedPermission
- Implementations of your own

2.1.1 StaticPermissions

A static permission states whether the user may perform an action or not. Consequently the only possible values are “true” and “false”.

Example: refreshButton=”true”

2.1.2 RoleBasedPermission

With a role-based permission, performing an action is tied to a role held by the user. Access to master data maintenance can be restricted to administrators in this way, for example.

A role-based permission is marked by the “#” character.

Examples: #admin; #gast; #manager.

2.1.3 FunctionBasedPermission

With a function-based permission, performing an action is tied to a function: the user has to be allowed to carry out one particular function. In the simplest case that means deleting, viewing, printing or adding a record.

Functions can be defined globally or for individual parts of an application. A user might be allowed to add records to the person table (user) while not holding that right for the client table.

A function-based permission is marked by the “$” character.

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

2.1.4 Implementations of your own

Besides the standard types, permission types of your own can be implemented.

2.1.5 Access control list

Permissions are always stated as an access control list (ACL): a semicolon-separated list of individual permissions. One list may mix different types of permission.

Example: $user.edit;#admin

If this list is assigned to the edit column of a ListControl, the column is shown only if the user is an administrator or holds the function “user.edit”.

Which permissions a user holds is recorded elsewhere — in a database or in a separate configuration file, for instance.

3 Implementing a permission system

The following steps are needed to use a permission system together with the Common Controls framework:

1. Providing the principal object
2. Registering the principal object
3. Connecting to a permission system
4. Configuring permissions within the control elements
5. Securing the action classes in struts-config.xml

3.1 Providing the principal object

If permissions are to be checked inside the control elements, the application has to provide a principal object. All it takes is to implement the Principal interface, which extends an existing class by the methods needed for checking permissions. That class may well be the user object created for a user at login, as in the examples below (see 3.3).

JAVA

```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.
        }
}

```

Code snippet 1: user object

3.2 Connecting to a permission system

Connecting to a permission system happens in the methods Principal#hasRight(String right) and Principal#isInRol(String role) and is the responsibility of the application. This is where an LDAP server or a user database can be queried, for example.

Online demo 2, for instance, uses a simple permission system configured through XML files, which is sufficient for small applications.

3.3 Registering the principal object

The class SecurityUtil provides the following methods for managing the principal object:

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

Table 1: The SecurityUtil class

The code example below shows how a user object is registered as the principal object once the user has been authenticated.

JAVA

```java
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 Exception {
        ctx.forwardToInput();
    }

    public void logon_onClick(FormActionContext ctx) throws Exception {

        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);
```

```
22
23        }
24  }
25
```

Code snippet 2: registering the principal object

A registered principal object is removed again by calling the static method unregisterPrincipal(), when the user logs off, for example.

CODE

```
1  SecurityUtil.unregisterPrincipal(ctx.session());
2
```

3.4 Configuring permissions

3.4.1 Control elements

Permissions can be assigned to control elements at the following levels:

Control element	
ListControl	- showing or hiding the control element itself - showing or hiding the add button - showing or hiding a column (for every type of column)
Tree	- showing or hiding the control element itself At node level no special mechanism is offered at present. The application therefore has to make sure itself that the data model contains only those nodes the user is allowed to see.
TreeListControl	- showing or hiding the control element itself - showing or hiding the add button - showing or hiding a column (for every type of column) At node level no special mechanism is offered at present. The application therefore has to make sure itself that the data model contains only those nodes the user is allowed to see.
TabSet	- showing or hiding the control element itself A tab of a tab set is disabled through the “enabled” attribute. If certain tabs are to be hidden altogether, the application still has to take care of that itself.
MenuControl	- showing or hiding the control element itself - showing or hiding a menu item

3.4.2 Tags

The Common Controls tag library provides two further tags with which a section of a JSP page can be executed depending on the user's permissions.

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

The <granted> tag executes the body of the tag only if the registered principal object holds the required permission.

JSP

```jsp
<%@ 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 executes the body of the tag only if the registered principal object does not hold the required permission.

JSP

```jsp
<%@ 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, too, function-based and role-based rights can be mixed freely.

3.5 Securing the action classes in struts-config.xml

By default the control elements generate hyperlinks that trigger an action on the server. The parameters needed for this are passed in the URL of the request.

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

URL generated when deleting the record with the id HOP:

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

URLs like this one can be typed into the address bar of a browser and executed by unauthorised users as well — whether or not the delete button was ever offered to them. The action triggered by the hyperlink therefore

has to be secured on the server side in its own right. The roles attribute in struts-config.xml serves that purpose; a role-based or function-based right can be assigned to it. The excerpt from a struts-config.xml file below shows how the permission check is extended to the execution of an action class.

Example: configuring permissions

CODE

```
<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>

```

So that the configured rights can be checked against the registered principal object, the following class also has to be entered as the RequestProcessor in the struts-config file.

CODE

```
<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 overrides the superclass's processRoles() method so that the roles attribute is checked against the registered principal object.

4 Examples

A detailed example of configuring permissions and using the principal object can be found in the samples accompanying online demo 2.

5 Abbreviations

ACL

Access control list

CC

Common Controls