Common-Controls Security

Version 1.6 - Stand: 14. Januar 2006

Common Controls

Herausgeber:
SCC Informationssysteme GmbH
64367 Mühltal

Tel: +49 (0) 6151 / 13 6 31 12
Internet http://www.scc-gmbh.com

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

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument beschreibt, wie sich Berechtigungen auf Ebene von Kontrollelementen vergeben lassen, um Programmfunktionen, auf bestimmte Anwenderkreise zu beschränken. Ein Kontrollelement verfügt hierzu über ein **permission**-Attribut, das eine Zuordnung von notwendigen Berechtigungen erlaubt (vgl. Abbildung 1). Dabei kann es sich um rollenbasierte oder funktionsbasierte Berechtigungen handeln. Die verschiedenen Berechtigungstypen werden im Abschnitt 2 dargestellt.

Das folgende Beispiel zeigt ein funktions basiertes Berechtigungssystem. Verfügt der Anwender nicht über die Rechte einen Datensatz anzulegen (create), zu bearbeiten (edit), zu löschen (delete) oder auszudrucken (print), werden ihm die entsprechenden Schaltflächen und Spalten nicht angezeigt (vgl. Abbildung 2).

```
<%@ 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>
```

Abbildung 1: Konfiguration von Berechtigungen für ein Kontrollelement

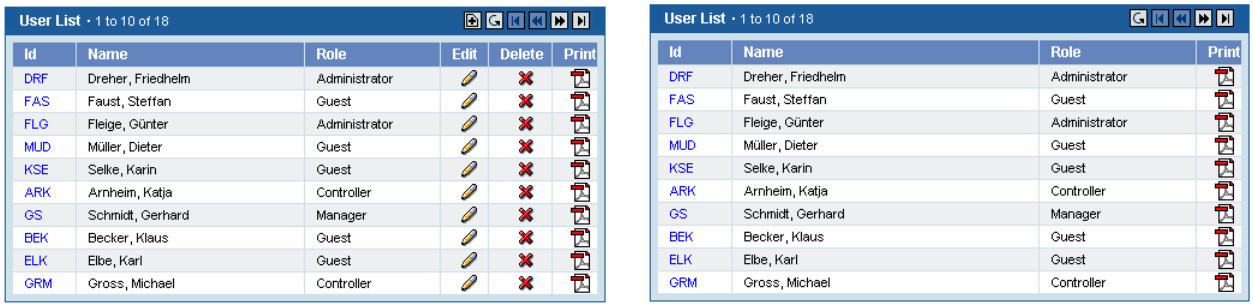


Abbildung 2: Anzeige ListControl bei unterschiedlicher Berechtigung

Das linke Bild zeigt die Liste wie sie einem Benutzer mit den Funktionsberechtigungen ***edit***,***delete***,***print*** und ***create*** angezeigt wird.
Der Benutzer auf der rechten Seite besitzt lediglich die Funktionsberechtigung ***print***.

2 Berechtigungstypen

Das Framework unterscheidet die folgenden Berechtigungen:

- StaticPermission
- RoleBasedPermission
- FunctionBasedPermission
- Eigene Implementierungen

2.1.1 StaticPermissions

Eine statische Berechtigung besagt, ob der Anwender eine Aktion durchführen darf oder nicht. Folglich können als Ausprägung nur die Werte „true“ oder „false“ vergeben werden.

Beispiel: refreshButton=“true“

2.1.2 RoleBasedPermission

Bei einer rollenbasierten Berechtigung ist die Ausführung einer Aktion an eine Rolle des Anwenders gebunden. Der Zugang zur Stammdatenpflege lässt sich so etwa generell auf Administratoren beschränken.

Die Kennzeichnung einer rollenbasierten Berechtigung erfolgt durch das „**#**“ Zeichen.

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

2.1.3 FunctionBasedPermission

Bei einer funktionsbasierten Berechtigung ist die Ausführung einer Aktion an eine Funktion gebunden. Der Anwender muss also eine spezielle Funktion ausführen dürfen. Im einfachsten Fall gehört hierzu etwa das Löschen, Ansehen, Drucken oder Hinzufügen eines Datensatzes.

Funktionen können dabei generell oder auch für einzelne Abschnitte innerhalb einer Anwendung definiert werden. So ist es etwa denkbar, dass ein Anwender in der Personentabelle (User) eventuell Datensätze hinzufügen kann; ihm dieses Recht aber in der Mandatentabelle (Client) nicht zusteht.

Die Kennzeichnung einer funktionsbasierten Berechtigung erfolgt durch das „**$**“ Zeichen.

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

2.1.4 Eigene Implementierungen

Neben den Standardtypen können auch eigene Berechtigungstypen realisiert werden.

2.1.5 Access Control List

Berechtigungen werden immer in Form einer Access Control List (**ACL**) angegeben. Dabei handelt es sich um eine Semikolon getrennte Liste mit Einzelberechtigungen. In einer Liste können dabei unterschiedliche Berechtigungstypen verwendet werden.

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

Hinterlegt man diese Liste mit Berechtigungen innerhalb eines ListControls auf der Edit-Spalte, so kann beispielsweise die Spalte nur dann anzeigen werden, wenn es sich bei dem Anwender um einen Administrator handelt oder dieser über die Funktion „user.edit“ verfügt.

Die Zuordnung von Berechtigungen zu einem Anwender erfolgt dabei etwa in einer Datenbank oder einer separaten Konfigurationsdatei.

3 Implementierung eines Berechtigungssystems

Die folgenden Schritte sind zur Nutzung eines Berechtigungssystem in Verbindung mit dem Common Controls Framework notwendig:

1. Bereitstellung des Principal Objektes
2. Registrierung des Principal Objektes
3. Anbindung an ein Berechtigungssystem
4. Konfiguration von Berechtigungen innerhalb der Kontrollelemente
5. Absicherung der Aktion Klassen in der struts-config.xml

3.1 Bereitstellung des Principal Objektes

Wenn eine Berechtigungsprüfung innerhalb der Kontrollelemente durchgeführt werden soll, muss die Applikation hierzu ein **`Principal Object`** zur Verfügung stellen. Hierzu wird lediglich das `Principal` Interface implementiert. Es erweitert eine bestehende Klasse um Methoden, die zur Überprüfung von Berechtigungen benötigt werden. Dabei kann es sich, wie in den folgenden Beispielen, etwa um das User Objekt handeln, welches für einen Anwender bei der Systemanmeldung generiert wird (dazu unter 3.3).

```
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 Anbindung an ein Berechtigungssystem

Die Anbindung an ein Berechtigungssystem erfolgt in den Methoden Principal#hasRight(String right) und Principal#isInRol(String role) und ist Aufgabe der Anwendung. Hier kann beispielsweise auf einen LDAP Server oder eine Benutzerdatenbank zugegriffen werden.

Die Online Demo 2 verwendet hier beispielsweise ein einfaches Berechtigungssystem welches sich über XML-Dateien konfigurieren lässt und für kleine Anwendungen ausreichend ist.

3.3 Registrierung des Principal Objektes

Für die Verwaltung des Principal Objektes stellt die Klasse **`SecurityUtil`** die folgenden Methoden zur Verfügung:

Methode	Beschreibung
`getPrincipal(...)`	Liefert das in der Session abgelegte Principal Objekt zurück.
`registerPrincipal(...)`	Registriert das Principal Objekt in der Session unter dem Schlüssel `Globals.PRINCIPAL_KEY`.
`unregisterPrincipal(...)`	Löscht das registrierte Principal Objekt aus der Session.

Tabelle 1: Die Klasse SecurityUtil

Das nachfolgende Code Beispiel zeigt die Registrierung eines User Objektes als Principal Objekt nach der Authentisierung des Anwenders.

```
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);

     }
}
```

CodeSnippet 2: Registrierung des Principal Objektes

Ein registriertes Principal Objekt wird durch den Aufruf der statischen Methode `unregisterPrincipal()` deregistriert (Beispielsweise beim Logoff des Anwenders).

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

3.4 Konfiguration von Berechtigungen

3.4.1 Kontrollelemente

Für Kontrollelemente können auf den folgenden Ebenen Berechtigungen vergeben werden:

Kontrollelement	
ListControl	- Kontrollelement selbst ein-oder ausblenden - Add-Button ein-oder ausblenden - Spalte ein-oder ausblenden (für alle Typen von Spalten)
Tree	- Kontrollelement selbst ein-oder ausblenden Auf Ebene der Knoten wird derzeit kein spezieller Mechanismus angeboten. Die Anwendung muss daher selbst sicherstellen, das das Datenmodell nur Knoten enthält, welche dem Anwender angezeigt werden dürfen.
TreeListControl	- Kontrollelement selbst ein-oder ausblenden - Add-Button ein-oder ausblenden - Spalte ein-oder ausblenden (für alle Typen von Spalten) Auf Ebene der Knoten wird derzeit kein spezieller Mechanismus angeboten. Die Anwendung muss daher selbst sicherstellen, das das Datenmodell nur Knoten enthält, welche dem Anwender angezeigt werden dürfen.
TabSet	- Kontrollelement selbst ein-oder ausblenden Eine Tabe eines Tabsets wird über das „enabled“-Attribut deaktiviert. Wenn bestimmte Taben nicht angezeigt werden sollen muss dies derzeit noch von der Applikation selbst sichergestellt werden.
MenuControl	- Kontrollelement selbst ein-oder ausblenden - Menuitem ein-oder ausblenden

3.4.2 Tags

Die Common Controls TagLibrary stellt darüber hinaus zwei weitere Tags zu Verfügung, mit denen sich die Ausführung eines Abschnittes in einer JSP Seite berechtigungsabhängig steuern lässt.

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

Das <granted> Tag führt den Inhalt des Tag-Bodies nur dann aus, wenn das registrierte Principal Objekt über die geforderte Berechtigung verfügt

```
<%@ 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>
```

Das <notGranted> Tag führt den Inhalt des Tag-Bodies nur dann aus, wenn das registrierte Principal Objekt **nicht** über die geforderte Berechtigung verfügt

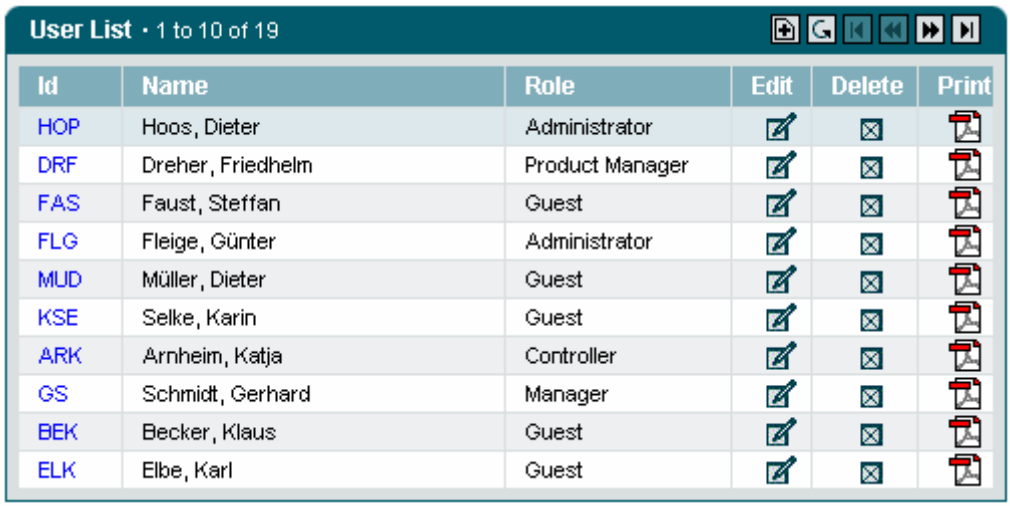
```
<%@ taglib uri="/WEB-INF/tlds/cc-security.tld" prefix="sec" %>

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

Funktions- und rollenbasierte Rechte lassen sich auch hier beliebig mischen.

3.5 Absicherung der Aktion Klassen in der struts-config.xml

Standardmäßig generieren die Kontrollelemente Hyperlinks, die zur Ausführung einer Aktion auf dem Server führen. Die dabei notwendigen Parameter werden in der URL innerhalb des Requests übermittelt.

Das folgende Beispiel zeigt dies für die „Delete“ Spalte einer Liste.



Generierte URL bei Löschen des Datensatzes mit der Id = HOP:

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

Solche URL’s lassen sich auch von unautorisierten Benutzern in die Adressleiste des Browsers eintragen und ausführen und zwar unabhängig davon, ob dem Anwender der Delete-Button angeboten wird. Folglich muss die Aktion, welche über den Hyperlink angestoßen wird selbst nochmals serverseitig abgesichert werden. Hierzu dient in der `struts-config.xml` Datei das **`roles`**-Attribut, zudem ein rollen- oder funktionsbasiertes Recht hinterlegt werden kann. Der nachfolgende Auszug aus einer `struts-config.xml` Datei zeigt, wie die Berechtigungsprüfung auf die Ausführung einer Aktion Klasse ausgedehnt wird.

Beispiel: Konfiguration von Berechtigungen

```
<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>
```

Damit die konfigurierten Rechte gegen das registrierte Principal Objekt geprüft werden können, muss in der `struts-config` Datei zudem die folgende Klasse als RequestProcessor eingetragen werden.

```
<struts-config>

      ...

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

      ...

</struts-config>
```

Die **FWRequestProcessor** Klasse ist von der Struts RequestProcessor Klasse abgeleitet und überschreibt die `processRoles`() Methode der Oberklasse, um die Prüfung des `roles` Attributes anhand des registrierten Principal Objekets zu ermöglichen.

4 Beispiele

Ein ausführliches Beispiel zur Konfiguration von Berechtigungen und der Verwendung des Principal Objektes findet sich in den Beispielen zur Online Demo 2.

5 Abkürzungsverzeichnis

A

ACL
Access Control List

C

CC
Common-Controls