Common-Controls Guided Tour FormTag's

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 Guided Tour FormTag's

1.1 Formulartypen

Mit den Common-Controls werden dem Pagedesigner Formulartypen geliefert, die immer wieder bei der Erstellung von Benutzeroberfläche benötigt werden. Durch die Verwendung der Formulartypen wird ein einheitliches Design innerhalb des Anwendung sichergestellt. Das Standarddesign kann auch an die eigenen Bedürfnisse (Corporate Identity) angepasst werden.

Folgende Formulartypen stehen derzeit zur Verfügung:

- Eingabeformular
- Anzeigeformular
- Formular für Fehler- und Erfolgsmeldungen
- Suchdialog
- Kopfzeile

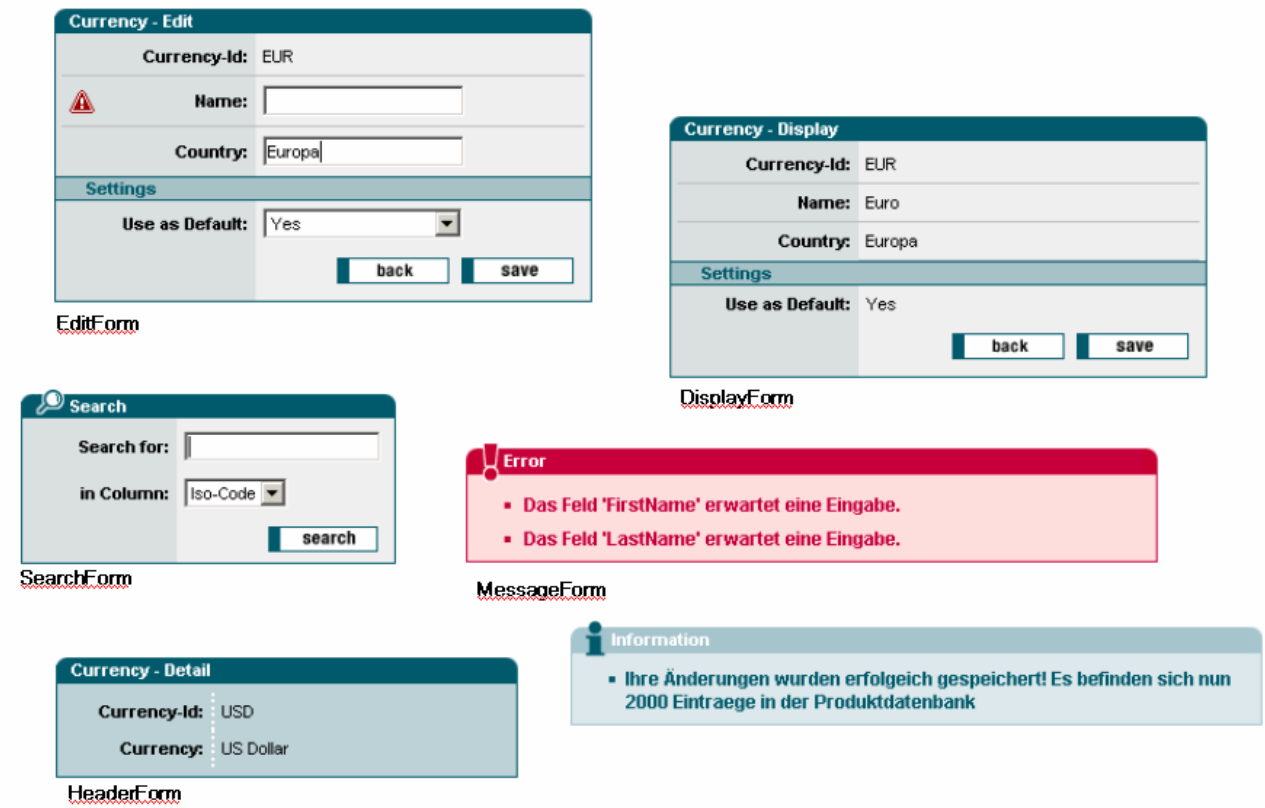
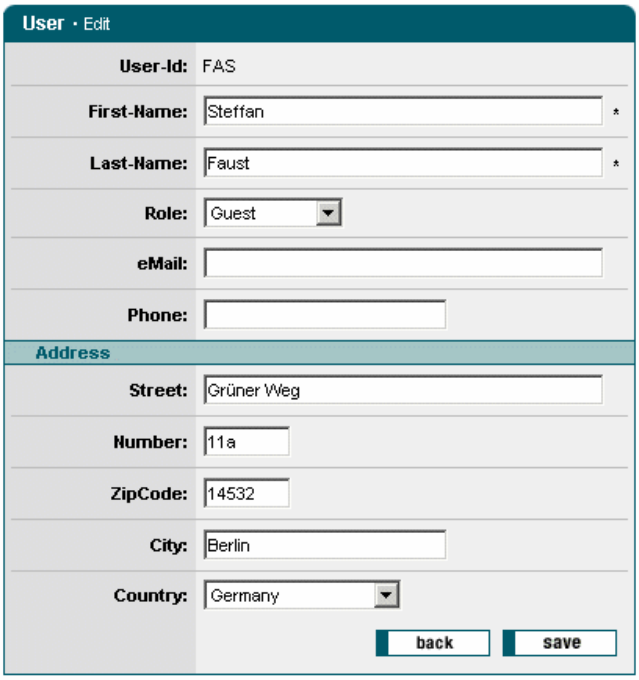


Abbildung 1: Formulartypen

1.2 Gegenstand

In dieser Übung erstellen wir ein Eingabeformular und implementieren zwei Callback-Methoden für den Back- und Save-Button. Das Formular enthält zwei Pflichtfelder, die bei einer fehlerhaften Validierung zu einer entsprechenden Meldung führen sollen.



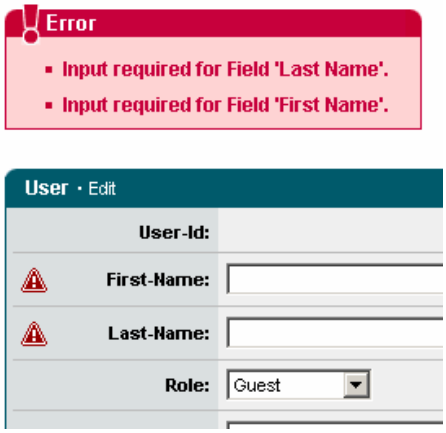


Abbildung 2: Eingabeformular

In dieser Übung werden folgende Punkte behandelt:

1. Auswahl des Designs für die Benutzeroberfläche
2. Erstellung der Actionklasse
3. Bereitstellung der Formulardaten
4. Festlegung des Formularaufbaus in der JSP-Seite
5. Implementierung der Callback-Methoden
6. Validierung und Fehlerpräsentation

1.3 Registrierung der Painterfactory

Zuerst erfolgt die Registrierung der Painterfactory. Sie legt fest, welches Design die Benutzeroberfläche erhält. Dies kann in der init()-Methode des Frontcontroler-Servlets applikationsweit geschehen.[1] Wir wählen hier das Standarddesign, welches uns der DefaultPainter bereitstellt.

```
import javax.servlet.ServletException

import org.apache.struts.action.ActionServlet;
import com.cc.framework.ui.painter.PainterFactory
import com.cc.framework.ui.painter.def.DefPainterFactory;
import com.cc.framework.ui.painter.html.HtmlPainterFactory;

public class MyFrontController extends ActionServlet {


    public void init() throws ServletException {

       super.init();

       // Register all Painter Factories with the preferred GUI-Layout
       // In this case we only use the Default-Layout.
       PainterFactory.registerApplicationPainter (
              getServletContext(), DefPainterFactory.instance());
       PainterFactory.registerApplicationPainter (
              getServletContext(), HtmlPainterFactory.instance());
    }
}
```

1.4 Ableitung der Action Klasse für den Struts-Adapter

In unserem Formular möchten wir die Detailinformationen zu einem Benutzer bearbeiten. Daher soll die Action-Klasse, welches das Befüllen unseres Formulares übernimmt, die Bezeichnung „UserEditAction“ tragen. Die Actionklasse wird dabei von der Klasse FWAction abgeleitet, welche die Struts-Action Klasse kapselt und um Funktionalitäten des Präsentationsframeworks erweitert. Dabei wird anstelle der execute()-Methode die doExecute()-Methode aufgerufen. [FWAction ist von org.apache.struts.action.Action abgeleitet] Sie erhält beim Aufruf den ActionContext, über den der Zugriff auf weitere Objekte, wie das Request- und Response-Objekt gekapselt ist.

```
import java.lang.IOException;

import com.cc.framework.adapter.struts.FWAction;
import com.cc.framework.adapter.struts.ActionContext;

public class UserEditAction extends FWAction {

    /**
     * @see com.cc.framework.adapter.struts.FWAction#doExecute(ActionContext)
     */
    public void doExecute(ActionContext ctx) throws Exception {
        // Code follows in the next chapter
    }
}
```

[1] Wenn der Benutzer zwischen verschiedenen Oberflächen wählen kann, werden zusätzliche PainterFactorys registriert. Dies erfolgt dann nicht innerhalb des Servlets, sondern in der LoginAction mit PainterFactory.registerSessionPainter() im Session Scope.

1.5 Bereitstellung der Formulardaten

Zum Befüllen unseres Formulares dient uns eine FormBean, das UserEditForm, welches direkt von org.apache.struts.action.ActionForm abgeleitet werden kann. Die FormBean wird in unserem Beispiel vereinfacht mit unserem UserObjekt initalisiert, das zuvor die Daten zu dem im Request übergebenen Schlüssel aus einer Datenbank geladen hat.

```
import java.lang.Exception;

import com.cc.framework.adapter.struts.FWAction;
import com.cc.framework.adapter.struts.ActionContext;

public class UserEditAction extends FWAction {

    /**
     * @see com.cc.framework.adapter.struts.FWAction#doExecute(ActionContext)
     */
    public void doExecute(ActionContext ctx) throws Exception {

        String userId = ctx.request().getParameter("userid");

        try {
            // Load the User
            User user = new User(userId);
            user.load();

            // Initialice the Form with the User-Data
            UserEditForm form = (UserEditForm) ctx.form();
            form.setUser(user);

            // In our Example we store the UserObject in our Session
            ctx.session().setAttribute("userobj", user);
        }
        catch (Throwable t) {
            ctx.addGlobalError("Error: ", t);
            ctx.forwardByName(Forwards.BACK);
        }

        // Call the JSP-Page with the Form
        ctx.forwardToInput();
    }

}
```

1.6 Festlegung des Formularaufbaus in der JSP-Seite

Um die Formular-Tags auf einer JSP Seite einzusetzen, muss am Anfang der Seite die entsprechende Tag Library deklariert werden. Anschließend können die Formularelemente mit dem Präfix <forms:*tagname* /> verwendet werden. [Zudem muss die Aufnahme der Tag Bibliothek im Deployment-Deskriptor, der WEB-INF/web.xml Datei, erfolgen].

Unser Formular enthält neben Eingabefeldern und Auswahlboxen auch einen Abschnitt zur Gruppierung von Informationen und eine Button-Leiste. Die benötigten Elemente werden über die folgenden Tags definiert:

Tag	Beschreibung
<forms:form/>	Definiert das Formular. Es wird die Überschrift festgelegt
<forms:plaintext/>	Dient zur Ausgabe von Text.
<forms:text/>	Erzeugt ein Eingabefeld. Zur Kennzeichnung von Pflichtfeldern dient das Required-Attribut.
<forms:select/>	Definiert eine Auswahlbox
<base:options/>	Definiert eine Optionsliste für eine Auswahlbox bereit
<forms:section/>	Definiert und zeichnet einen Absatz
<forms:buttonsection/>	Definiert eine Button-Leiste und legt einen Default-Button fest
<forms:button/>	Zeichnet einen Buttton.
<forms:message/>	Zeichnet einen Meldungsdialog. Über das Attribut severity="error" wird der Dialog als Fehlerdialog klassifiziert.

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

<forms:message severity="error" caption="Error" formid="err"/>

<html:form action="/sample101/userEdit">
    <forms:form type="edit" caption="User - Edit" formid="frmEdit">
        <forms:plaintext
            label="User-Id"
            property="userId"/>

        <forms:text
            label="First-Name"
            property="lastName"
            size="45"
            required="true"/>

        <forms:text
            label="Last-Name"
            property="firstName"
            size="45"
            required="true"/>

        <forms:select   label="Role"    property="rolekey">
            <base:options property="roleOptions"/>
        </forms:select>

        <forms:text
            label="eMail"
            property="email"
            size="45"/>

        <forms:text
            label="Phone"
            property="phone"
            size="25" />

        <forms:section title="Address">
            <forms:text
                label="Street"
                property="street"
                size="45"/>
            <forms:text
```

```
                label="Number"
                property="streetnumber"
                size="5"/>
            <forms:text
                label="ZipCode"
                property="zipcode"
                size="5"/>
            <forms:text
                label="City"
                property="city"
                size="25"/>

            <forms:select   label="Country"     property="countrycode">
                <ctrl:options
                    property="countryOptions"
                    labelProperty="country"/>
            </forms:select>
        </forms:section>

        <forms:buttonsection default="btnSave">
            <forms:button name="btnBack" src="btnBack1.gif"/>
            <forms:button name="btnSave" src="btnSave1.gif"/>
        </forms:buttonsection>
    </forms:form>
</html:form>
```

Das <form>-Tag ist in unserem Beispiel in ein Struts <html:form>-Tag eingebettet. Es erhält damit über die angegebene Action (/sample101/userEdit) den Zugriff auf die Form-Bean, welche die Anzeigedaten bereitstellt. Das <form>-Tag kann auch alleine ohne Struts <html:form> Tag verwendet werden, dann muss jedoch zusätzlich das action-Attribut angegeben werden.

Alle Tags der Common Controls Bibliothek arbeiten bei Bedarf im Verbund mit den Struts Tags!

1.7 Implementierung der Callback-Methoden

Der Back- und Save-Button in unserem Formular erzeugen jeweils ein Click-Event, auf das wir innerhalb unserer Action durch die Aufnahme zweier Callback-Methoden reagieren können. Der Name der Methode setzt sich dabei aus dem Namen der Schaltfläche und dem Suffix **onClick** zusammen. Formularbuttons müssen dabei mit dem Prefix „**btn**“ benannt werden. Ansonst wird keine Callback-Methode aufgerufen.

Der Button **btnBack** führt damit zum Aufruf der Methode **back_onClick**. Der Methode wird dabei der FormActionContext übergeben, welcher den Zugriff auf das Request-, Session Objekt und die FormBean kapselt.

```
import java.lang.Exception;

import com.cc.framework.adapter.struts.FWAction;
import com.cc.framework.adapter.struts.ActionContext;
import com.cc.framework.adapter.struts.FormActionContext;

public class UserEditAction extends FWAction {

    /**
     * @see com.cc.framework.adapter.struts.FWAction#doExecute(ActionContext)
     */
    public void doExecute(ActionContext ctx) throws Exception {
         // Code see above
    }

    // ------------------------------------------------
    //               Event Handler
    // ------------------------------------------------

    /**
     * This Method is called when the Back-Button is pressed.
     * @param  ctx  FormActionContext
     */
    public void back_onClick(FormActionContext ctx) throws Exception {
         ctx.forwardByName(Forwards.BACK);
    }

    /**
     * This Method is called when the Save-Button is pressed.
     * @param  ctx  FormActionContext
     */
    public void save_onClick(FormActionContext ctx) throws Exception {
         // See next Chapter
    }

}
```

1.8 Validierung und Fehlerpräsentation

Die Validierung der Daten erfolgt in unserem Beispiel mittels der validate()-Methode in der FormBean. Die Methode wird innerhalb der UserEditAction aufgerufen, sobald der Save-Button auf unserem Formular angeklickt wurde. Das Formular kann vor dem Feld, in dem ein Fehler aufgetreten ist einen entsprechenden visuellen Hinweis generieren. Hierzu wird die Fehlermeldung unter Angabe des entspechenden Properties in die ActionErrors-Collection eingestellt.

Die nachfolgende Validierung führt zu der Meldung „Input required for Field: First Name“ und zeigt ein Warnzeichen for dem entsprechenden Feld an, wenn dort keine Eingabe erfolgt ist.

```
import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionMapping;

public class UserEditForm extends UserDisplayForm {

   /**
    * @see org.apache.struts.action.ActionForm#validate()
    */
   public ActionErrors validate(ActionMapping mapping,
         HttpServletRequest request) {

         ActionErrors errors = new ActionErrors();

         if ("".equals(firstName) ) {
               errors.add("firstName",
                     new ActionError("Input Required for Field: ", "First Name"));
         }

         if ("".equals(lastName) ) {
               errors.add("lastName",
                     new ActionError("Input Required for Field: ", "Last Name"));
         }

         return errors;
   }

}
```

Die Validierung wird in der save-onClick()-Methode angestoßen. Auftretende Fehler werden dabei in den **FormActionContext** eingestellt. Dies führt dazu, dass nach einer Rückkehr auf die Eingabeseite die entsprechenden Fehlermeldungen angezeigt werden.

Wenn die Validierung erfolgreich war, können die Änderungen in die Datenbank übernommen werden. Bei diesem Vorgang können ebenfalls Fehler auftreten. Solche Fehler werden in unserem Beispiel nicht in der Eingabemaske angezeigt, sondern in der Maske, von der aus wir in den Bearbeitungsmodus verzweigt sind. Die Fehlermeldung wird dazu ebenfalls in den Kontext eingestellt und dann anschließend die entsprechende Action aufgerufen. Die CommonControls Trail-Version enthält dazu den ausführlichen Source-Code.

Im Erfolgsfall wird eine entsprechende Meldung an den Anwender weitergegeben. Der Text wird hierzu über die Methode addGlobalMessage() in den FormActionContext eingestellt. Anschließend wird die Eingabemaske wieder verlassen.

```
import java.lang.Exception;

import com.cc.framework.adapter.struts.ActionContext;
import com.cc.framework.adapter.struts.FWAction;
import com.cc.framework.adapter.struts.FormActionContext;

public class UserEditAction extends FWAction {

    // other code see above ...

    public void save_onClick(FormActionContext ctx) throws Exception {

          UserEditForm form = (UserEditForm) ctx.form();

          // Validate the Formdata
          ActionErrors errors = form.validate(ctx.mapping(), ctx.request());
          ctx.addErrors(errors);

          // If there are any Errors return and display a Message
          if (ctx.hasErrors()) {
               ctx.forwardToInput();
               return;
          }

          try {
               // In our Example we get the User-Object from the Session
               User user = (User) ctx.session().getAttribute("userobj");
               populateBusinessObject(ctx, user);
               user.update();
          }
          catch (Throwable t) {
               ctx.addGlobalError("Error: ", t);
               ctx.forwardByName(Forwards.BACK);
               return;
          }

          // Generate a Success Message
          ctx.addGlobalMessage("Data updated: ", form.getUserName());

          ctx.forwardByName(Forwards.SUCCESS);
    }

}
```

1.9 Tour Ende

Das Beispiel hat gezeigt, wie sich durch den Einsatz von Form-Tag's Oberflächen schneller erstellen lassen. Dabei wird die Einhaltung eines einheitlichen Designs innerhalb der Anwendung gewährleistet. Es lassen sich aber auch andere Designs durch eine Anpassung der Painter realisieren. Dabei können verschiedene Designs parallel verwendet werden.

Features Form-Tag's:

- Bereitstellung von zahlreichen Formularelementen (Text, Plaintext, Textarea, Select, Button, Buttonsection, File, Password, Radio, Spin, Description, Checkbox, Section)
- Gruppierungsmöglichkeit von Formularelementen
- Formularelemente können einfach mit einem Label, Beschreibungstext, Erforderliche Eingabe etc. in der JSP-Seite deklariert werden.
- Abbildung von Formular Ergeignissen auf Event-Handler in der Action Klasse
- Visualisierung bei Fehleingaben.
- Unterstützung von Hover-Effekten bei Buttons
- Design des Formulares in der JSP-Seite oder auch serverseitig definierbar.
- Design durch Painterfactory an eigenen StyleGuide (Corporate Identity) anpassbar.
- Gleiches Look and Feel in Microsoft InternetExplorer > 5.x und Netscape Navigator > 7.x

1.10 Exkurs Bereitstellung von Buttons mit Hover-Effekt

Zur Darstellung des Hover-Effektes, wird ein Button im aktiven Zustand und ein Button im selektierten Zustand benötigt. Der aktive Button wird dabei als gif-Datei mit dem Präfix **btn** und dem Suffix **1.gif** abgelegt (z.B. btnBack1.gif). Für den Hover-Effekt wird ein Button mit der Endung **3.gif** benötigt (z.B. btnBack3.gif).

Namenskonventionen und Zustände für Formualbutton:

Zustand	Beispiel	Namenskonvention
Aktiv		btnXXX1.gif
Inaktiv		btnXXX2.gif
HOver		btnXXX3.gif
Pressed		btnXXX4.gif

Die Bilder werden automatisch ausgetauscht, sobald der Mauszeiger über einen Formularbutton bewegt wird. Zuständig hierfür ist ein JavaScript Eventhandler, der für das MouseOver und MouseOut-Event in der Datei fw/def/jscript/controls.js registriert wird.

Dieses Script wird von dem Default Painter automatisch in jede HTML Seite inkludiert[2].

[2] Dies geschieht im Beispiel in der Schablonendatei jsp\template\Main.jsp, welche von allen JSP-Seiten für das Seitenlayout verwendet wird. Mit der Zeile <util:jsp directive="includes"/> werden hier alle notwendigen Includes des Frameworkes eingebunden.

2 Glossar

C

CC

Common-Controls