Common-Controls Guided Tour TreeListControl

Version 1.0.3 - Last changed: 01. August 2004

Common Controls

Herausgeber:
SCC Informationssysteme GmbH
64367 Mühltal

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

Product Site:
www.common-controls.com

Table of contents

1 Guided Tour TreeListControl

1.1 Object

This exercise demonstrates the use of the TreeListControls. It combines a tree with a list, whose nodes can be opened and closed. For this, the programmer merely provides the display data (of the data model) by the implementation of a simple interface.

The TreeListControl offers the following features:

- The lines at the uppermost level can be displayed or hidden. Different images can be stored in an ImageMap for the nodes and the leaves. The assignment of the images to the relevant tree nodes takes place with the help of regular expressions.
- The control element independently administers all the necessary status data across several Server Roundtrips. This includes, for example, the exploded or closed status of a tree node.
- Check boxes can be displayed or hidden before the tree entries. When selecting a node at a lower level, all the higher-level nodes are selected automatically.

The following steps are required for using the TreeListControl:

- Selection of the layout for the user interface.
- Derivation of an action class.
- Instancing of a TreeListControl.
- Provision of display data.
- Configuration of the control element within the JSP-Page.

1.2 Registration of the Painterfactory

The registration of the Painterfactory takes place first. It defines which design the user interface will get. This can be done across the application in the init()-method of the Frontcontroler-Servlet.[1] Here, we select the standard design that the DefaultPainter offers us.[2]

```
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-Design
       // In this case we use the Default-Design.
       PainterFactory.registerApplicationPainter (
              getServletContext (), DefPainterFactory.instance());
       PainterFactory.registerApplicationPainter (
              getServletContext (), HtmlPainterFactory.instance());
    }
}
```

1.3 Derivation of the action class

In our TreeListControl, we want to display regions and Federal states. Therefore, the action class which takes care of the loading and filling of the TreeListControl must have the nomenclature "RegionBrowseAction".

The action class is then derived from the class FWAction, which encapsulates the Struts-action class and extends with functionalities of the presentation framework. Instead of the execute()-method, the doExecute()-method is called. [FWAction is derived from org.apache.struts.Action]. On calling, it contains the ActionContext, through which the access to additional objects such as the Request- and Response-object is capsulated

```
import java.io.IOException;
import javax.servlet.ServletException;

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

public class RegionBrowseAction extends FWAction {

    /**
     * @see com.cc.framework.adapter.struts.FWAction#doExecute(ActionContext)
     */
    public void doExecute(ActionContext ctx)
        throws IOException, ServletException {
        // In the next chapter, we will instantiate
        // our TreeListControls with the DisplayData
    }
}
```

[1] If it has to be possible for the individual user to choose between different interface designs, then additional PainterFactorys are registered in the user session. This is done mostly in the LoginAction with PainterFactory.registerSessionPainter() in the session Scope.

[2] Additional designs (PainterFactories) are included in the kit of the Professional Edition, or you can develop them yourself.

1.4 Instancing of the TreeListControl

Now, the TreeListControl is instanced within our action and filled with the display data. The data model is assigned to the control element through the setDataModel()-method. The method takes, as the argument, an object of type **TreeGroupDataModel** . What this involves is an interface which provides access to the display data of the tree. It is the job of the application developer to provide a corresponding implementation.

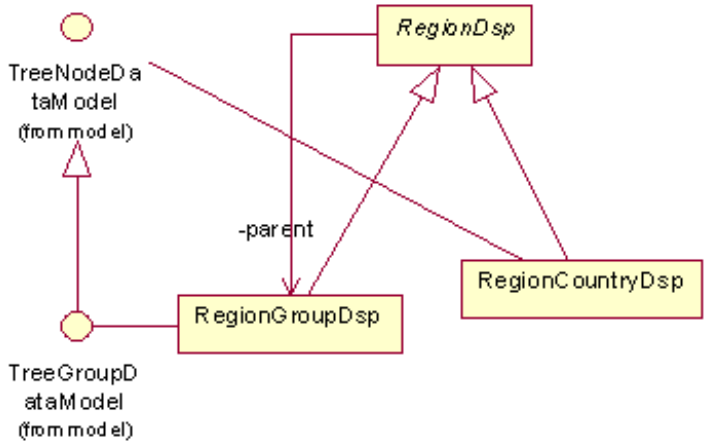
```
import java.io.IOException;
import javax.servlet.ServletException;

import com.cc.framework.adapter.struts.ActionContext;
import com.cc.framework.adapter.struts.FWAction;
import com.cc.framework.ui.control.TreeListControl;

public class RegionBrowseAction extends FWAction {

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

        try {
            // first get the Displaydata for our TreeList
            RegionGroupDsp dspData = DBRegion.fetchDspOutline();

            // Create the TreeListControl an populate it
            // withe the Data to display
            TreelistControl regionList = new TreelistControl();
            regionList.setDataModel(dspData);

            // third put the TreeListControl into the Session-Object.
            // Our TreeListControl is a statefull Object.
            // Normaly  you can use an Objectmanager or an other
            // workflow Component that manage the Livecyle of the Object
            ctx.session().setAttribute("regions", regionList);
        }

        catch (Throwable t) {
            ctx.addGlobalError("Error: ", t);
        }

        // Display the Page with the TreeList
        ctx.forwardToInput();
    }
}
```

1.5 Provision of the display data

The tree consists of group and leaf nodes. Group nodes can, in turn have additional nodes (composite pattern). Accordingly, for both the node types, the interfaces **TreeGroupDataModel** and **TreeNodeDataModel** are available (TreeGroupDataModel extendsTreeNodeDatamodel). With their help, the tree structure can be easily generated.

In doing so, the root node is generated first and under it, additional groups or leaves are suspended. The root node is passed onto the TreeControl as a data model.

The procedure corresponds to the provision of display data for the TreeControl. In contrast to the TreeControl, however, the TreeListControl should present additional columns. To do so, our Bean, which provides the display data, must merely implement additional properties for the corresponding columns. In our example, what is involved is the class RegionDsp, from which group and leaf nodes are derived.



A detailed code example will be provided to you with the trial version, which you can download free of cost.

1.6 Configuration of the TreeControls within the JSP-Page

In order to use the TreeListControl-tag on a JSP page, the corresponding tag library must be declared at the start of the page. Then, the Common-Controls can be referenced with the prefix <ctrl:*tagname* />. [In addition, the tag libraries must be included in the Deployment descriptor, the WEB-INF/web.xml file]

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

<ctrl:treelist
    id="tl1"
    name="regions"
    action="sample301/regionBrowse"
    title="Regions Structure"
    rows="15"
    refreshButton="true"
    expandMode="multiple"
    root="true">

    <ctrl:columntree
        title="Region"
        property="region"
        width="180"
        imageProperty="type"/>

    <ctrl:columntext
        title="Name"
        property="name"
        width="250"/>

    <ctrl:columnadd
        title="Add"
        property="add"/>

    <ctrl:columnedit
        title="Edit"
        property="editable"/>

    <ctrl:columndelete
        title="Delete"
        property="editable"/>
</ctrl:treelist>
```

Since we have saved the TreeListControl in the session, the name of the Bean is specified via the Name attribute. In addition, with the action attribute, that action is specified to which actions from our TreeListControl (onEdit, onDelete, etc...) are delegated.

If the TreeListControl is saved in a FormBean, it is enough to specify the property attribute. In this case, the scope of the Formbean must be set to "session", so that the control element can retain its state across server roundtrips.

When using a workflow control, the control element can be generated with other components and later deleted from the session.

All the necessary steps for using the TreeListControls are thus commplete. The opening and closing behavior does not have to be self-implemented. It is administered by the control element itself. For navigation, the control element provides scrolling buttons which are enabled as soon as the specified number of lines is exceeded.

1.7 Tour End

The TreeListControl can be easily and quickly integrated. Its standard behavior can also be overwritten if required. Thus, it is also possible to generate special TreeListList objects, which already encapsulate the access to certain technical data and can be repeatedly used within an application project.

Thanks to the configuration options in the JSP-Page, the behavior of the TreeListControls can be changed quickly. Alternative designs can be easily integrated by customizing the existing Painter. Parallel support to different designs is also possible. The programmer can concentrate on the technical sequences and on providing the display data.

Features of the TreeListControl:

- Implements automatic exploding and closing of nodes.
- Administers the state of optional checkboxes.
- Different configuration options (display/hiding of the root node, opening and closing behavior is modiable, connecting lines can be suppressed at the highest level).
- Data below a group node can also be loaded only when the group is opened. The tree does not have to be fully known right from the beginning. This is e.g. very helpful in conjunction with databases. When a node with an unknown number of children is first exploded, an onExpandEx event is sent to the application.
- Design of the TreeListControl can be defined in the JSP or also on the server side!
- Maps actions that are carried out on the tree to CallBack methods in the action class (Examples: onCheck, onExpand, onCollapse, onExpandEx).
- Images from the nodes/leaves assignable with regular expressions.
- Authorization check at node level. Nodes can thus be automatically suppressed for unauthorized users. (see Scurity Documentation).
- Design can be customized with Painterfactory to own StyleGuide (Corporate Identity).
- Optimized HTML-Code.
- The same Look-and-Feel in Microsoft InternetExplorer > 5.x and Netscape Navigator > 7.x

1.8 Exkurs: Implementation of Callback methods

The TreeListControl automatically generates an onDrilldown-Event on clicking on a label, to which the programmer can react within the action class.

To react to this event in our example, we include a corresponding Callback method in the RegionBrowseAction. Since we do not wish to implement the business logic here, we forward the event to another action - RegionDisplayAction.

```
import java.io.IOException;
import javax.servlet.ServletException;

import com.cc.framework.adapter.struts.ActionContext;
import com.cc.framework.adapter.struts.FWAction;
import com.cc.framework.ui.control.TreeListControl;

public class RegionBrowseAction extends FWAction {

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

        try {
             RegionGroupDsp dspData = DBRegion.fetchDspOutline();

             TreelistControl regionList = new TreelistControl();
             regionList.setDataModel(dspData);

             ctx.session().setAttribute("regions", regionList);
        }

        catch (Throwable t) {
             ctx.addGlobalError("Error: ", t);
        }

        // Display the Page with the TreeList
        ctx.forwardToInput();
    }

      // ------------------------------------------------
      //          event handler
      // ------------------------------------------------

      /**
       * This Method is called when the TreeLabel is clicked
       * In our Example we switch to the DetailView, which shows
       * more Information about the node.
       * @param    ctx    ControlActionContext
       * @param    key    UniqueKey, as created in the Datamodel
       */
      public void regions_onDrilldown(ControlActionContext ctx, String key) {
             ctx.forwardByName(Forwards.DRILLDOWN, key);
      }
}
```

The name of the CallBack method is composed of the property name of the TreeListControls - the name of the Bean - and the event that has occurred. Since the TreeListControl was stored under the name "region" in the session, the name of the CallBack-method is **regions_**onDrilldown.

1.9 Alternative Layouts

Other layouts can be generated by means of the implementation and registration of own Painterfactorys. Here are a few examples from application projects.

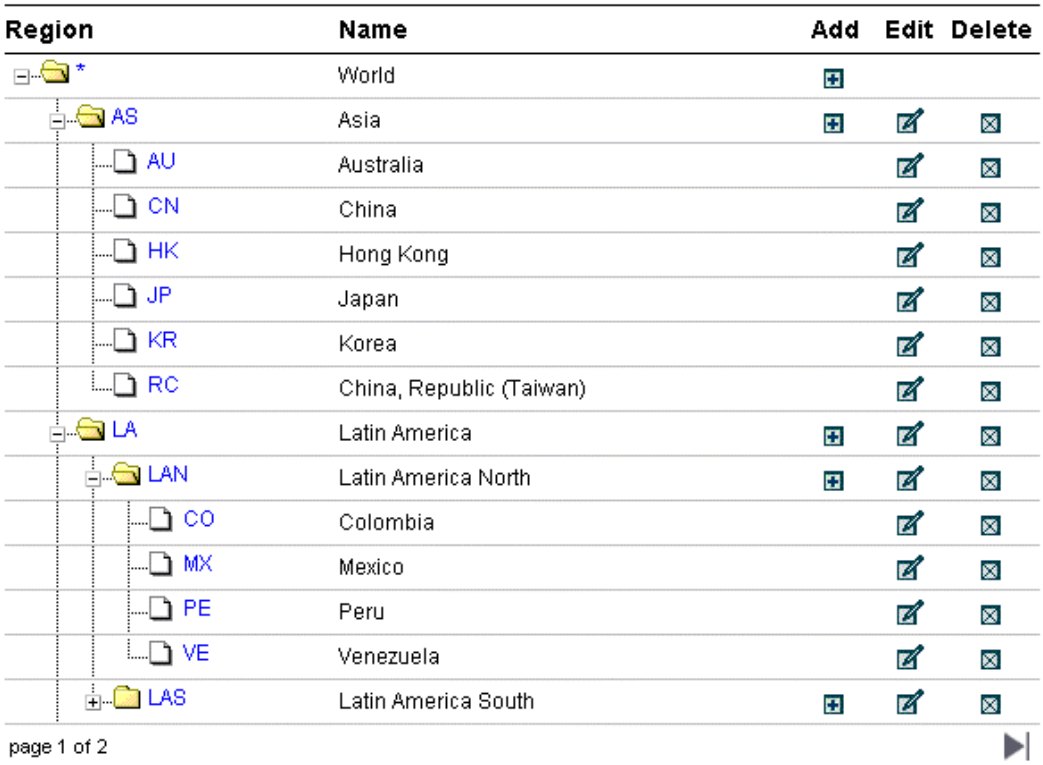


Figure 1: Example Layout 1

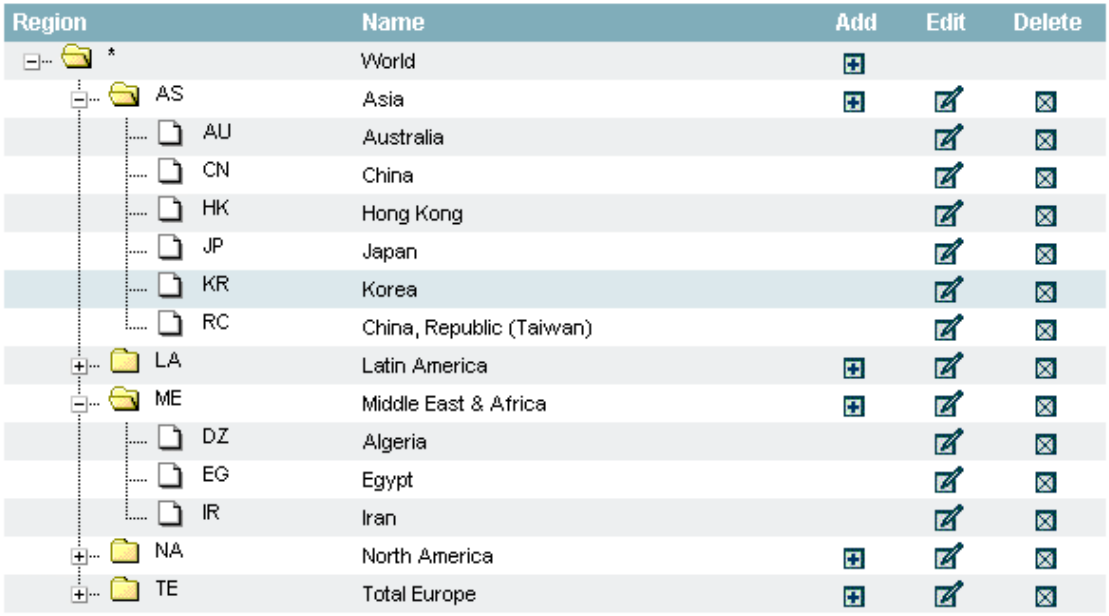


Figure 2: Example Layout 2

2 Glossary

C

CC

Common-Controls