Common-Controls
Guided Tour
TreeControl

Version 1.0.3 - Last changed: 01. August 2004

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 TreeControl

1.1 *Object*

This exercise demonstrates the use of the TreeControl. This control element generates a tree whose nodes can be exploded and closed. For this, the programmer merely provides the display data (of the data model) by the implementation of a simple interface.

TreeControl 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.

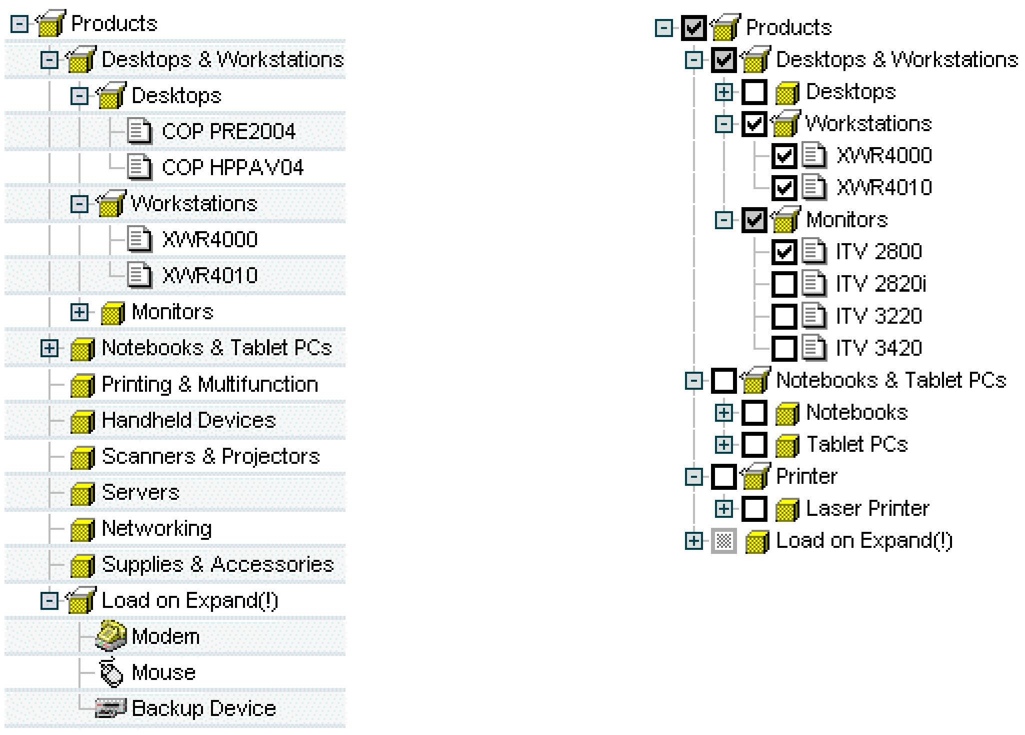


Figure 1: Example TreeControl

Only the following steps are required for using the TreeControl:

1. Selection of the design for the user interface.
2. Creation of an action class.
3. Instancing of a TreeControl.
4. Provision of display data.
5. Configuration of the tree 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 for the Struts adapter

In our tree, we want to display product groups and products. Therefore, the action class which takes care of the loading and filling of the TreeControl must have the nomenclature "ProductTreeBrowseAction".

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. On calling, it gets the ActionContext, through which the access to additional objects such as the Request- Session- 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 ProductTreeBrowseAction 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 TreeControls 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 TreeControl

Now, the TreeControl 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.TreeControl;

public class ProductTreeBrowseAction extends FWAction {

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

          try {
               // first we get the Data for our Tree
               ProductGroupDsp data = DBProduct.fetch();

               // secondly create the TreeControl and populate it
               // with the Data to display
               TreeControl products = new TreeControl();
               products.setDataModel(data);

               // third put the TreeControl into the Session-Object.
               // Our Control is a statefull Object.
               ctx.session().setAttribute("products", products);
          }

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

          // Display the Page with the Tree
          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.

```
// Root
ProductGroupDsp root = new ProductGroupDsp("0", "Products", "Root");

ProductGroupDsp group = null;
ProductGroupDsp subgroup = null;

// First Group under the Root-Element
group = new ProductGroupDsp("1201", "Workstations & Monitors");

subgroup = new ProductGroupDsp("2102", "Workstations");
subgroup.addChild( new ProductDsp("3005", "XWR4000", "product description"));
subgroup.addChild( new ProductDsp("3005", "XWR4010", "product description"));
group.addChild(subgroup);

subgroup = new ProductGroupDsp("2103", "Monitors");
subgroup.addChild( new ProductDsp("3101", "ITV 2800") );
subgroup.addChild( new ProductDsp("3102", "ITV 2820i") );
subgroup.addChild( new ProductDsp("3103", "ITV 3220") );
group.addChild(subgroup);

root.addChild(group);

// Secound Group under the Root-Element
group = new ProductGroupDsp("1204", "Printing & Multifunction");
root.addChild(group);
```

The class ProductGroupDsp

```
public class ProductGroupDsp extends ProductBaseDsp implements TreeGroupDataModel {

     /**
      * ParentNode
      */
     private TreeGroupDataModel parent = null;

     /**
      * ChildNodes
      */
     private Vector children = new Vector();

     // ------------------------------------------------
     //                Methods
     // ------------------------------------------------

     /**
      * Constructor
      * @param  key               Unique Key for the Group
      * @param  name              Name of the Group
      * @param  unknownChildren   true if the Childs should be loaded later
      *                           false if the Childnodes exists
      */
     public ProductGroupDsp(String key, String name) {
          super();

          this.key = key;
          this.name = name;
          this.type = "prodgroup";
     }

     /**
      * Constructor
      * @param  key               Unique Key for the Group
      * @param  name              Name of the Group
      * @param  description       description for the Group
      * @param  unknownChildren   true if the Childs should be loaded later
      *                           false if the Childnodes exists
      */
     public ProductGroupDsp(String key, String name, String description) {
          super();

          this.key = key;
          this.name = name;
          this.description = description;
          this.type = "prodgroup";
     }

     /**
      * @see TreeGroupDataModel#getChild(int)
      */
     public TreeNodeDataModel getChild(int index) {
          return (TreeNodeDataModel) children.elementAt(index);
     }
     /**
      * @see TreeGroupDataModel#addChild(TreeNodeDataModel)
      */
     public void addChild(TreeNodeDataModel child) {

          children.add(child);

          child.setParent(this);
     }

     /**
      * Returns the Number of ChildNodes
      * -1 = The Number of ChildNodes is unknown.
      *      When the Node opens an onExpandEx Event is generated
      *      and the Childs can be loaded at runtime
      * 0  = This Node has no ChildNodes
      * >0 = This Node has ChildNodes
      *
      * @see TreeGroupDataModel#size()
      */
     public int size() {
          return children.size();
     }

     /**
      * @see TreeNodeDataModel#getParent()
      */
     public TreeGroupDataModel getParent() {
          return parent;
     }

     /**
      * @see TreeNodeDataModel#setParent(TreeGroupDataModel)
      */
     public void setParent(TreeGroupDataModel parent) {
          this.parent = parent;
     }

     /**
      * @see TreeNodeDataModel#getParentKey()
      */
     public String getParentKey() {
          return parent.getUniqueKey();
     }

     /**
      * @see TreeNodeDataModel#getUniqueKey()
      */
     public String getUniqueKey() {
          return this.key;
     }
}
```

The class ProductDsp

```
public class ProductDsp extends ProductBaseDsp implements TreeNodeDataModel {

     /**
      * ParentNode
      */
     private TreeGroupDataModel parent = null;

     /**
      * Constructor
      * @param     key    Unique Productkey
      * @param     name   Productname
      */
     public ProductDsp(String key, String name) {
          super();

          this.key = key;
          this.name = name;
          this.type = "product";
     }

     /**
      * Constructor
      * @param     key          Unique Productkey
      * @param     name         Productname
      * @param     description  Product description
      */
     public ProductDsp(String key, String name, String description) {
          super();

          this.key = key;
          this.name = name;
          this.description = description;
          this.type = "product";
     }

     public void setParent(TreeGroupDataModel parent) {
          this.parent = parent;
     }

     public TreeGroupDataModel getParent() { return parent; }
     public String getParentKey() { return parent.getUniqueKey(); }
     public String getUniqueKey() { return this.key; }

}
```

The class ProductBaseDsp

```
public class ProductBaseDsp {

       /**
        * ProductKey
        */
       protected String key = "";

       /**
        * Name of the Product
        */
       protected String name = "";

       /**
        * Description for the Product
        */
       protected String description = "";

       /**
        * Type for the Node
        */
       protected String type = "";


       /**
        * Constructor
        */
       public ProductBaseDsp() {
              super();
       }

       public String getName() { return name; }
       public String getDescription() { return description; }
       public String getType() { return type; }

}
```

1.6 Configuration of the TreeControls within the JSP-Page

In order to use the TreeControl on a JSP page, the corresponding tag library must be declared at the start of the page. Then, the Common-Controls can be used 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:tree
       id="prodtree1"
       name="products"
       action="sample201/productBrowse"
       root="true"
       linesAtRoot="true"
       labelProperty="name"
       imageProperty="type"
       expandMode="multiple"
       groupselect="true"
       checkboxes="false"/>
```

All the necessary steps for using the TreeControl are thus complete.
The opening and closing behavior does not have to be self-implemented. It is administered by the control element itself. This also applies to the selection states of checkboxes, which can be activated with the attribute checkboxes="true". The programmer can thus concentrate on the technical sequences and on providing the display data.

Professional Edition: With the attribute runat="client", the tree is generated in a JavaScript version and and can thus be exploded and closed without Server Roundtrips. However, this increase in the user comfort does not need any changes in the application program.

1.7 Tour End

The TreeControl can be easily and quickly integrated. Its standard behavior can also be overwritten if required. Thus, it is also possible to generate special Tree- 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 TreeControl can be changed quickly. Alternative designs can be easily integrated by customizing the existing Painter. Parallel support to different designs is also possible.

Features of the TreeControl:

- 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 TreeControl can be defined in the JSP or also on the server side!
- Maps the action that is to be carried out on the tree to CallBack methods in the action class (Examples: onCheck, onExpand, onCollapse, onExpandEx).
- Images can be assigned in front of the nodes/leaves through regular expressions.
- Authorization check at node level. Nodes can thus be automatically hidden from unauthorized users. (see Security Documentation).
- Layout through Painterfactory can be customized to own StyleGuide (Corporate Identity).
- Optimized HTML-Code.
- Same Look and Feel in Microsoft® InternetExplorer > 5.x and Netscape™ Navigator > 7.x

1.8 Exkurs: Implementation of a Callback method

The TreeControl 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 ProductTreeBrowseAction. Since we do not wish to implement the business logic here, we forward the event to another action - ProductDisplayAction.

```java
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.ControlActionContext;
import com.cc.framework.ui.control.TreeControl;

import com.cc.sampleapp.common.Forwards;

public class ProductTreeBrowseAction extends FWAction {

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

          try {
               ProductGroupDsp data = DBProduct.fetch();
               TreeControl products = new TreeControl();
               products.setDataModel(data);
               ctx.session().setAttribute("products", products);
          }

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

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

      // ------------------------------------------------
      //          Tree-Control 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  products_onDrilldown(ControlActionContext ctx, String key) {
            ctx.forwardByName(Forwards.DRILLDOWN, key);
      }
}
```

The name of the CallBack is composed of the Property name of the TreeControl - the name of the Bean - and the event that has occurred. Since the TreeControl was saved in the Session under the name "products", the name of the CallBack method is **products**_onDrilldown.

1.9 Exkurs: Use of an ImageMap

The TreeControl uses predefined images for depicting exploded and closed nodes; these images can be easily replaced if required. A separate image can be assigned to every entry within the tree.

One option for this is the use of an ImagMap. The ImageMap is declared outside the tree in the JSP-Page and assigned to the TreeControl with the attribute **imagemap**.

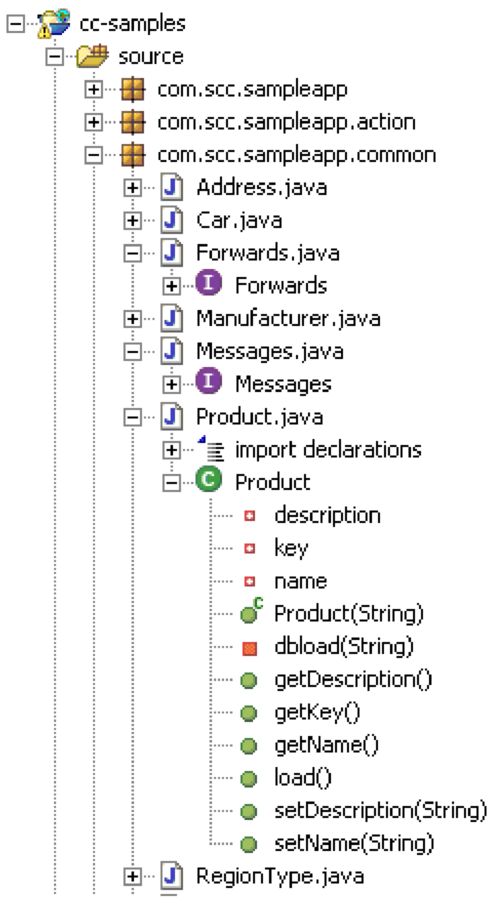
```
<%@ taglib uri="/WEB-INF/tlds/cc-controls.tld" prefix="ctrl" %>
<%@ taglib uri="/WEB-INF/tlds/cc-utility.tld"  prefix="util" %>

<util:imagemap name="imap_products">
    <util:imagemapping
        rule="prodgroup.open"
        src="images/imgBoxOpen.gif"
        width="16" height="16"/>
    <util:imagemapping
        rule="prodgroup.closed"
        src="images/imgBoxClosed.gif"
        width="16" height="16"/>
    <util:imagemapping
        rule="product"
        src=" images/imgItem.gif"
        width="16" height="16"/>
</util:imagemap>

<ctrl:tree
    id="prodtree1"
    name="products"
    action="sample201/productBrowse"
    root="true"
    linesAtRoot="true"
    labelProperty="name"
    imageProperty="type"
    imagemap="imap_products"
    expandMode="multiple"
    groupselect="true"
    checkboxes="true"/>
```

Working:
When depicting the tree, the data model returns one expression each via the method getType(), which is compared with the ImageMap. In case of a tally with a rule, the corresponding image is drawn. For closed and exploded nodes, these expressions are automatically extended with the suffix ".open" or ".closed", so that it is possible to use different images for different states. The rules are specified with regular expressions.

The method that returns the expression for the image to be drawn is determined using the attribute **imageProperty**. In our example, that type-property is used, which always returns "prodgroup" for groups and "product" for individual leaves.

1.10 Alternative Layouts

Other layouts can be generated by means of the implementation and registration of own Painterfactory, as the following example shows.



```
cc-samples
  source
    com.scc.sampleapp
    com.scc.sampleapp.action
    com.scc.sampleapp.common
      Address.java
      Car.java
      Forwards.java
        Forwards
      Manufacturer.java
      Messages.java
        Messages
      Product.java
        import declarations
        Product
          description
          key
          name
          Product(String)
          dbload(String)
          getDescription()
          getKey()
          getName()
          load()
          setDescription(String)
          setName(String)
      RegionType.java
```

2 Glossary

C

CC	Common-Controls