SCC
INFORMATIONSSYSTEME

COMMON CONTROLS · GUIDED TOUR

TreeControl

Adding the control element to an existing application, step by step

SCC Informationssysteme GmbH

Version 1.6 · As of: 14.01.2006

Impressum

PUBLISHED BY

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

Contents

1 Guided Tour TreeControl

1.1 Object

This tour demonstrates the use of the TreeControl. The control element produces a tree whose nodes can be expanded and collapsed. All the programmer has to provide is the display data - the data model - by implementing a simple interface.

The TreeControl offers the following features:

- The lines on the topmost level can be shown or hidden. Nodes and leaves can carry different images, held in an image map. Images are assigned to a tree node by regular expression.
- The control element keeps track of all the state it needs across several server round trips - which nodes are expanded, for instance.
- Check boxes can be shown in front of the tree entries. Selecting a node on a lower level automatically marks all the nodes above it.

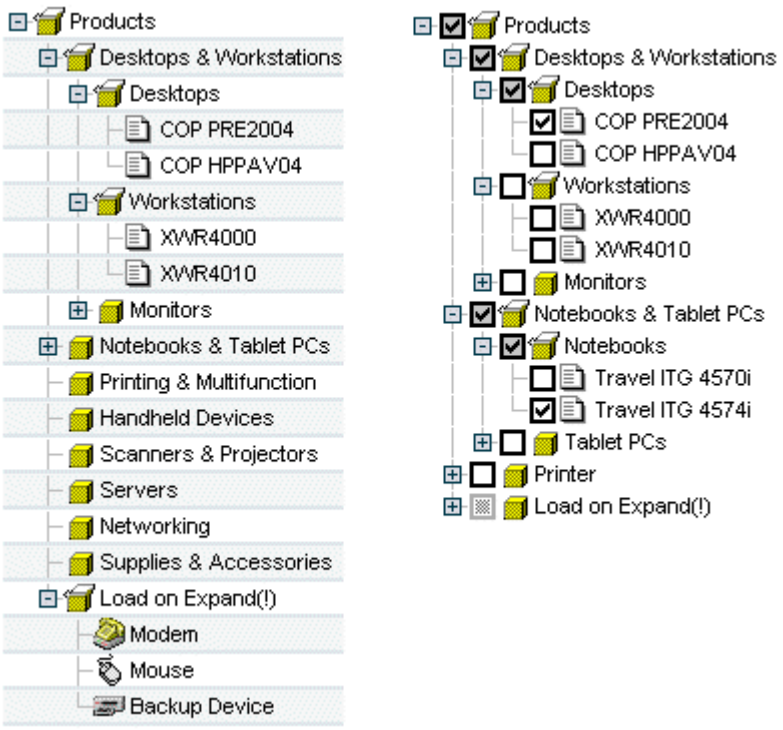

Figure 1: TreeControl example

Using the TreeControl takes no more than the following steps:

1. Choosing the design of the user interface
2. Writing an action class
3. Instantiating a TreeControl
4. Providing the display data
5. Configuring the tree within the JSP page

1.2 Registering the painter factory

The first step is to register the painter factory. It determines the design of the user interface. This can be done for the whole application in the init() method of the front controller servlet.[1] Here we choose the standard design provided by the DefaultPainter.[2]

JAVA

```java
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] If individual users are to choose between different interface designs, additional painter factories are registered in the user session. This is usually done in the LoginAction with PainterFactory.registerSessionPainter() in session scope.

[2] Further designs (painter factories) are part of the Professional Edition, or you develop your own.

1.3 Deriving the action class for the Struts adapter

Our tree is to show product groups and products, so the action class that loads and fills the TreeControl is called "ProductTreeBrowseAction". It is derived from the class FWAction, which wraps the Struts action class and adds the functions of the presentation framework. Instead of execute(), the doExecute() method is called. It receives the ActionContext, which wraps the access to further objects such as the request, session and response object.

JAVA

```java
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
    }
}
```

1.4 Instantiating the TreeControl

The TreeControl is now instantiated within our action and filled with the display data. The data model is assigned to the control element through the setDataModel() method, which takes an object of type TreeGroupDataModel. That is an interface providing access to the display data of the tree; supplying an implementation of it is the job of the application developer.

JAVA

```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.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 Providing the display data

The tree consists of group nodes and leaf nodes. Group nodes can hold further nodes in turn (composite pattern). Accordingly there is one interface per node type, TreeGroupDataModel and TreeNodeDataModel (TreeGroupDataModel extends TreeNodeDataModel). Together they make building the tree structure straightforward.

The root node is created first, and further groups or leaves are hooked in below it. The root node is then handed to the TreeControl as its data model.

JAVA

```java
// 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 following classes make up the models for this example:

- ProductGroupDsp
- ProductDsp
- ProductBaseDsp

1.6 Configuring the TreeControl within the JSP page

To use the TreeControl tag on a JSP page, the corresponding tag library has to be declared at the top of the page. The Common Controls are then available with the prefix <ctrl:tagname />. [The tag libraries also have to be listed in the deployment descriptor, the file WEB-INF/web.xml]

JSP

```jsp
<%@ 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"/>
```

That covers every step needed to use the TreeControl. Expanding and collapsing does not have to be implemented, the control element handles it. The same goes for the selection state of check boxes, which are switched on with the attribute checkboxes="true". The programmer can concentrate on the business logic and on providing the display data.

Professional Edition: with the attribute runat="client" the tree is generated as a JavaScript version and can then be expanded and collapsed without server round trips. This gain in comfort takes no changes at all in the application program.

1.7 Tour end

The TreeControl is quick and simple to integrate, and its standard behaviour can be overridden where needed. That allows special TreeControl objects which already wrap the access to particular business data and can be reused throughout an application project.

The configuration options in the JSP page make it quick to change how the TreeControl behaves. Alternative designs are a matter of adapting the existing painters, and several designs are supported side by side.

Features of the TreeControl:

- Expanding and collapsing nodes happens automatically.
- Keeps track of the state of optional check boxes.
- Various configuration options (showing or hiding the root node, changing the expand and collapse behaviour, hiding the connecting lines on the topmost level).
- Data below a group node can be loaded only when the group is opened, so the tree does not have to be known in full from the start - useful in combination with a database. When a node with an unknown number of children is expanded for the first time, the application receives an onExpandEx event.
- The design of the TreeControl can be defined in the JSP page or on the server side.
- Maps actions performed on the tree to callback methods in the action class (for example onCheck, onExpand, onCollapse, onExpandEx).
- Images in front of nodes and leaves are assigned by regular expression.
- Permission check at node level, so nodes are hidden automatically from users without the required permission (see the security documentation).
- The layout can be adapted to your own style guide (corporate identity) through a painter factory.
- Compact HTML code.
- Same look and feel in Microsoft® Internet Explorer > 5.x and Netscape™ Navigator > 7.x

1.8 Excursus: implementing a callback method

When a label is clicked, the TreeControl automatically raises an onDrilldown event, which the programmer can react to within the action class.

To react to that event in our example, we add a matching callback method to the ProductTreeBrowseAction. Since we do not want to implement the business logic at this point, we pass the event on to another action, ProductDisplayAction.

JAVA

```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 method is made up of the property name of the TreeControl - the name of the bean - and the event that occurred. Since the TreeControl was put into the session under the name "products", the callback method is called products_onDrilldown.

1.9 Excursus: using an image map

For open and closed nodes the TreeControl uses predefined images, which are easy to replace. Every entry in the tree can be given an image of its own, and one way to do that is an image map. It is declared outside the tree in the JSP page and assigned to the TreeControl through the attribute "imagemap".

JSP

```
<%@ 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"/>
```

How it works: while the tree is drawn, the data model returns an expression through the method getType() for each entry, and that expression is matched against the image map. Where a rule matches, the corresponding image is drawn. For closed and open nodes the expression is automatically extended by the suffix ".closed" or ".open", so each state can use a different image. The rules themselves are regular expressions.

The method that returns the expression for the image to be drawn is named in the attribute "imageProperty". Our example uses the type property, which returns "prodgroup" for groups and "product" for individual leaves.

1.10 Alternative layouts

Other layouts are produced by implementing and registering painter factories of your own, as the following example shows:

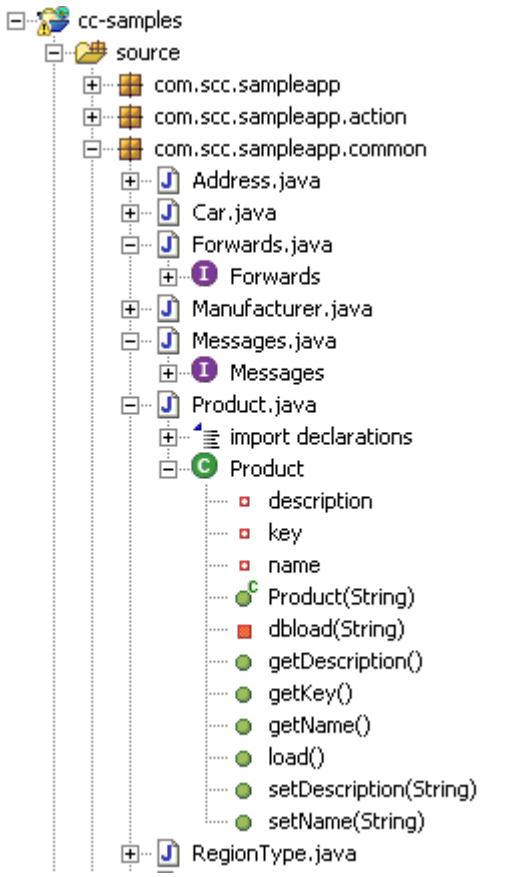

Figure 2: Layout examples using a custom painter factory

2 Glossary

CC	Common Controls
JSP	JavaServer Pages
Painter	Renders the HTML code of a control element; a custom painter factory adapts the layout to your corporate design.
DataModel	Interface through which a control element obtains its display data.