JAVAWORLD
SOLUTIONS FOR JAVA DEVELOPERS

This story appeared on JavaWorld at
http://www.javaworld.com/javaworld/jw-09-2007/jw-09-checkboxtree.html

Swing-based tree layouts with CheckboxTree

A configurable tree component with checkable nodes

By Lorenzo Bigagli and Enrico Boldrini, JavaWorld.com, 09/11/07

CheckboxTree is an open source, Swing-based tree component with a checkbox in each of its nodes, similar to those commonly found in installers but missing from the Swing GUI toolkit. In this article creators Lorenzo Bigagli and Enrico Boldrini introduce CheckboxTree and demonstrate its standout features, namely four configurable check-propagation styles, grayed checkboxes, and a custom renderer that allows you to display radio buttons rather than checkboxes in your tree layouts. The article includes the source code for CheckboxTree, which you may use or extend for your Swing GUI development projects.

Installation GUIs and application preference windows often feature checkbox-enabled tree components, but you won't find such a component in the Swing GUI toolkit. Most developers either end up extending the Swing `JTree` ad-hoc or using one of the tree extensions available. We actually tried several of these components for a recent project but found that none of them had quite the features, simplicity or flexibility we needed.

Some components were part of an overly complicated widget library or required extending specific classes that disrupted our class hierarchy design. Some were not open source or relied on libraries that were not open source. Some had native dependencies. Most important, we needed a component that would support several checkbox propagation styles in a tree layout. We also wanted grayable checkboxes to indicate if descendants of a given node were in the opposite checking state from that node. So, like many developers before us, we created our `CheckboxTree` component from scratch.

`CheckboxTree` is a Swing `JTree` component with a checkbox in each of its nodes, as shown in Figure 1 (click the image for a live demonstration).

Figure 1. CheckboxTree: Checked, selected and grayed paths are visible

In this article we introduce `CheckboxTree` and explain its relationship to the Swing `JTree` classes. We also discuss its architecture and provide implementation details. Finally, we highlight some of the current limitations of `CheckboxTree` and note improvements that could be made to it in the future. `CheckboxTree` is available under the GPL license, so you are free to use it in your Swing development projects.

Background: JTree concepts and terminology

Before we begin describing `CheckboxTree` in detail, it might be helpful to quickly recall some Swing `JTree` concepts and terminology:

- **JTree**, as described in the Swing documentation, is a "control that displays a set of hierarchical data as an outline."
- A **node** is the basic unit of data displayed by a `JTree`. A specific node can be identified either by a `TreePath` or by its display row.
- **TreePath** is an object that contains a `JTree` node and all its ancestors.
- **TreeModel** is an object that manages the `JTree` data model based on the well-known MVC design pattern.
- **TreeSelectionModel** is an object that manages user selections in a `JTree`, typically rendered by highlighting selected nodes.
- **TreeCellRenderer** is used to style `JTree` nodes according to their content and status.

Using CheckboxTree

A key feature of `CheckboxTree` is that it does not require you to use tree nodes that implement a specific "checkable-node" interface (e.g., an interface with a method `isChecked`). You can continue to work with your preferred tree model, such as the default `TreeModel` or a custom one, which makes it easier to integrate `CheckboxTree` into your existing code.

You can use `CheckboxTree` by invoking one of the provided constructors, which are modeled after the standard `JTree` ones. The sample below shows how you would construct a `CheckboxTree` with a default `TreeModel`.

```
CheckboxTree checkboxTree = new CheckboxTree();
```

Here is how you would construct a `CheckboxTree` from an existing `TreeNode`:

```
TreeNode yourRoot = new DefaultMutableTreeNode("foo"); CheckboxTree checkboxTree = new CheckboxTree(yourRoot);
```

Here is how you would construct a `CheckboxTree` from an existing `TreeModel`:

```
Model yourTreeModel = new DefaultTreeModel(new DefaultMutableTreeNode("foo")); CheckboxTree checkboxTree = new CheckboxTree(yourTreeModel);
```

You could also set the data model at a later time, as shown here:

```
Model yourNewTreeModel = new DefaultTreeModel(new DefaultMutableTreeNode("bar")); checkboxTree.setModel(yourNewTreeModel);
```

Once constructed, your `CheckboxTree` is ready to listen for user input and respond.

State management and event handling

All `CheckboxTree` nodes include a checkbox. If the checkbox is enabled, the user can check it to indicate his or her selection. A *checked node* is a tree node whose checkbox has been checked; in the same way, a *checked path* is a `TreePath` whose node has been checked. In `CheckboxTree` we use the term *checking* to indicate the set of paths that are checked at a given point in time, just as in `JTtree` the term *selection* indicates the set of paths that are selected.

Going back to Figure 1 and playing with the applet, you will notice that the check events on a node may propagate to its descendants and/or ancestors, according to the check-propagation style selected. We'll discuss check propagation below. You will also notice that some checkboxes have a grayed background: that means that the checkbox of at least one descendant node is in the opposite state from that node. On the other hand, if a checkbox has a white background, you can be sure that the checkboxes of all its descendants are in the same state. `CheckboxTree` uses a `TreeCheckingModel` to maintain the checking and background consistency. This is similar to how a `JTree` maintains selection consistency by means of a `TreeSelectionModel`.

Some useful commands

Here is how you can retrieve the list of checked paths:

```
TreePath[] tp = checkboxTree.getCheckingPaths();
```

If you are only interested in retrieving the *roots* of your checked subtrees, you can use the following:

```
TreePath[] tp = checkboxTree.getCheckingRoots();
```

Here is how you would register a `TreeCheckingListener` to listen for changes to the checking state of your `CheckboxTree`.

```
checkboxTree.addTreeCheckingListener(new TreeCheckingListener() {
      public void valueChanged(TreeCheckingEvent e) {
          System.out.println("Checked paths changed: user clicked on "
              + (e.getLeadingPath().getLastPathComponent()));
      }
});
```

Customizing CheckboxTree

So far you have seen the basic usage of a `CheckboxTree`. In this section we'll introduce two ways to customize this component. First, we'll show you how to configure the component's check propagation style. Then, we'll show you how to write a customized `CheckboxTreeCellRenderer`, which will allow you to display your favorite checkbox control in a tree layout.

Configurable checking modes

In `CheckboxTree`, *checking mode* describes the way a check event is propagated to other checkboxes. We have implemented four checking modes for the component, which can be set as shown in Listing 1. (Note that you can also add new modes by implementing your own `TreeCheckingMode` class.)

Listing 1. Checking modes in CheckboxTree

```
 checkboxTree.getCheckingModel().setCheckingMode(TreeCheckingModel.Check
ingMode.SIMPLE);

checkboxTree.getCheckingModel().setCheckingMode(TreeCheckingModel.Check
ingMode.PROPAGATE);

checkboxTree.getCheckingModel().setCheckingMode(TreeCheckingModel.Check
ingMode.PROPAGATE_PRESERVING_CHECK);

CheckboxTree.getCheckingModel().setCheckingMode(TreeCheckingModel.Check
ingMode.PROPAGATE_PRESERVING_UNCHECK);
```

Each of the above styles uses different rules to propagate a check event. The checking model takes care of the background accordingly. The modes are as follows:

- **Simple** toggles the just-clicked checkbox only.
- **Propagate** toggles the just-clicked checkbox and propagates the change down. In other words, if the clicked checkbox is checked all the descendants will be checked; otherwise all the descendants will be unchecked.
- **Propagate preserving check** propagates the change not only to descendants but also to ancestors. With regard to *descendants* this mode behaves exactly like the Propagate mode. With regard to *ancestors* it checks/unchecks them as needed so that a node is checked if and only if all of its children are checked.
- **Propagate preserving uncheck** propagates the change not only to descendants but also to ancestors. With regard to *descendants* this mode behaves exactly like the Propagate mode. With regard to *ancestors* it checks/unchecks them as needed so that a node is unchecked if and only if all of its children are unchecked.

Checkbox rendering

`CheckboxTree`'s `DefaultCheckboxTreeCellRenderer` renders a tree node using a textual label and a special checkbox that supports grayed background rendering. It is also possible to customize checkbox rendering by implementing the `CheckboxTreeCellRenderer` interface.

This interface extends the Swing `TreeCellRenderer` interface in two ways: First, it re-declares the `getTreeCellRendererComponent`, just as a reminder for the implementor to properly display the checking/grayed status of the node. We would have preferred to enforce this in some other way, such as by adding a method parameter, but that would have required changes to the Swing classes, which we considered impractical.

Second, it declares the method `isOnHotspot(int x, int y)`, which returns whether the specified relative coordinates insist on the intended checkbox control. This information may be used by a mouse listener to determine whether to toggle a node or not. In fact, it allows `CheckboxTree` to accommodate an arbitrary `CheckboxTreeCellRenderer`.

Example: A custom CheckboxTreeCellRenderer

Figure 2 shows a `CheckboxTree` that has been customized to display radio buttons rather than checkboxes. Other custom renderers could use their own custom checkboxes rather than `CheckboxTree`'s quadristate checkbox. It is possible to customize a `CheckboxTreeCellRenderer` in many different ways. For instance, in one project we put in the `CheckboxTree` objects belonging to different classes. A custom `CheckboxTreeCellRenderer` then showed a different icon as needed.

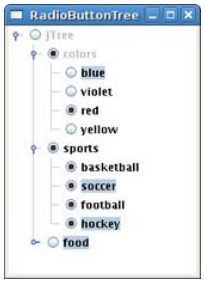

Figure 2. A customized CheckboxTreeCellRenderer

Listing 2 shows the `RadioButtonTree` implementation. You could use this code as a starting point to write a new customized renderer.

Listing 2. The RadioButtonTree

```
public class RadioButtonTreeCellRenderer implements
CheckboxTreeCellRenderer {

   JRadioButton button = new JRadioButton();
   JPanel panel = new JPanel();
   JLabel label = new JLabel();

   public boolean isOnHotspot(int x, int y) {
     return (button.getBounds().contains(x, y));
   }

   public RadioButtonTreeCellRenderer() {
     label.setFocusable(true);
     label.setOpaque(true);
     panel.setLayout(new FlowLayout(FlowLayout.LEFT, 0, 0));
     panel.add(button);
     panel.add(label);
     button.setBackground(UIManager.getColor("Tree.textBackground"));
     panel.setBackground(UIManager.getColor("Tree.textBackground"));
   }

   public Component getTreeCellRendererComponent(JTree tree, Object
value, boolean selected, boolean expanded, boolean leaf, int row,
boolean hasFocus) {
     label.setText(value.toString());
     if (selected)

label.setBackground(UIManager.getColor("Tree.selectionBackground"));
     else
       label.setBackground(UIManager.getColor("Tree.textBackground"));
     TreeCheckingModel checkingModel = ((CheckboxTree)
tree).getCheckingModel();
     TreePath path = tree.getPathForRow(row);
     boolean enabled = checkingModel.isPathEnabled(path);
     boolean checked = checkingModel.isPathChecked(path);
     boolean grayed = checkingModel.isPathGreyed(path);
     button.setEnabled(enabled);
     if (grayed) {
       label.setForeground(Color.lightGray);
     } else {
       label.setForeground(Color.black);
     }
     button.setSelected(checked);
     return panel;
   }

   public static void main(String[] args) {
     CheckboxTree tree = new CheckboxTree();
     tree.getCheckingModel().setCheckingMode(CheckingMode.SIMPLE);
     tree.setCellRenderer(new RadioButtonTreeCellRenderer());
     JFrame frame = new JFrame("RadioButton tree");
     frame.add(tree);
     tree.expandAll();
     frame.pack();
     frame.setVisible(true);
   }
}
```

Inside CheckboxTree

`CheckboxTree` extends the Swing `JTree` with features for visualizing and managing checkboxes and checking events. We created a specialized `TreeCheckingModel` class to manage checking events in `CheckboxTree`. Our `TreeCheckingModel` has an API similar to `TreeSelectionModel`, which is the class that manages selection events for Swing GUI components. The `TreeCheckingModel` API is easy to use for any developer familiar with the Swing GUI toolkit. The similarity between our `TreeCheckingModel` and the Swing `TreeSelectionModel` extends to the class hierarchy and inheritance structure, as you can see in Figure 3.

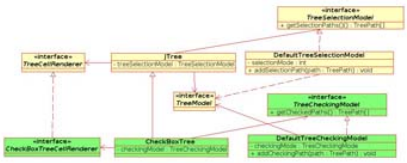

Figure 3. TreeSelectionModel (yellow classes) and TreeCheckingModel (green classes) compared; click for a larger image

You will note that the `JTree` class uses a `TreeModel`, a `TreeCellRenderer`, and a `TreeSelectionModel` to carry out its functions. The `CheckboxTree` component extends `JTree` to also use the `TreeCheckingModel` API. Moreover, `CheckboxTree` requires an implementation of `CheckboxTreeCellRenderer`, which is a `TreeCellRenderer` that displays a checkbox or similar components inside each tree node (we provide a `DefaultCheckboxTreeCellRenderer` with a `QuadristateCheckbox`).

We needed a `TreeCellRenderer` capable of displaying a checkbox in four states: checked, unchecked, gray checked, and gray unchecked. For this we implemented a QuadristateCheckbox, which is able to support a GUI's given look and feel and display the four states. States are managed by a model called the `QuadristateButtonModel`. The class hierarchy for the `QuadristateButtonModel` is shown in Figure 4. The `QuadristateCheckbox` is actually a hack that allows us to display two new states within a normal `JCheckbox`. Our hack is adapted from a similar one created by Dr. Heinz M. Kabutz for his TristateCheckBox.

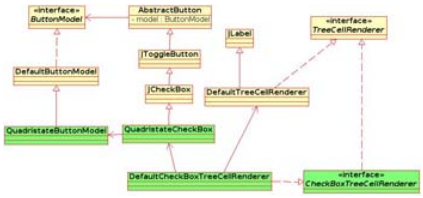

Figure 4. Class diagram of the QuadristateCheckbox

The future of CheckboxTree

We plan to fix some known limitations to `CheckboxTree` in future releases. The main limitation has to do with the component's performance when dealing with very big data models. For example, if a user clicks on a node with many descendants, the GUI freezes for the time required to insert all the affected subpaths in the relevant `TreeCheckingModel` data structures. We plan to fix this by optimizing the memorization of the descendants' checking state. Only the ancestors will need to be stored in an appropriate set, which will greatly reduce execution time.

Another limitation concerns the rendering of gray states in `CheckboxTree` checkboxes, which is done by setting the armed property to true in the model. The price of this is that we can't show roll-over events on the checkboxes. A better implementation of the `QuadristateCheckbox` will allow us to manage roll-over events and display them appropriately.

Another rendering problem occurs when the checkbox position inside the renderer is relative to the path to be displayed. Swing design calls for a unique renderer object that is used by every row. Thus we can't rely on it to know if the click has be made inside a checkbox, when different rows have checkboxes at different positions. If you have ideas about fixing any of these problems, or ideas for new features, please feel free to contribute!

In conclusion

In this article we have introduced `CheckboxTree`, a handy new widget for your Swing GUI development. While other `JTree` components have been extended to include checkboxes, we believe `CheckboxTree` offers unique features and flexibility. We've explained how `CheckboxTree` extends from the Swing `JTree` widget, introduced its unique, configurable propagation model and grayed checkboxes, and highlighted some of the implementation details that could make it easier for you to use and extend `CheckboxTree`.

See the Resources section to learn more about the Swing GUI toolkit, Swing development, and other components that extend the functionality of the Swing `JTree`. Join the discussion that starts at the bottom of this page to let us know about your experiences with Swing `JTree` components -- especially `CheckboxTree`!

About the author

Lorenzo Bigagli is a researcher at the Italian National Research Council and the University of Florence, currently taking a PhD on advanced technologies for Spatial Data Infrastructures. He has loved Java for more than 10 years (and counting), particularly for its support for concurrent programming and networking.

Enrico Boldrini is a computer scientist currently taking a MSc at the University of Florence. He also works for the Italian National Research Council (in the Earth Space Information Technologies Laboratory) developing Java applications for Geomatics.

Read more about Core Java in JavaWorld's Core Java section.

JavaWorld's Daily Brew
Starting conversations in the Java developer community

Performance

Submitted by Anonymous on Mon, 12/07/2009 - 11:07.

Tags: Core Java Java Development Tools Swing/GUI Programming

Back to:
www.javaworld.com/javaworld/jw-09-2007/jw-09-checkboxtree.html.

The design of this component is really great, especially with regards to the fact that it doesn't require you to use the TreeNode interface when you need dig-in mode (as opposed to the similar JIDE component). It supports your custom TreeModel right out of the box!

However, the performance is far from good enough. Selecting a node with about 2000 child nodes in dig-in mode takes about 20 seconds - which is far from what one could call a responsive component.

Average:
Your rating: None

Login to post comments Email this page Printer-friendly version

Having checkbox only for parent nodes

Submitted by gpshankar on Wed, 06/29/2011 - 01:04.

Hi,
with this checkboxtree, is it possible to customize it for having check boxes only for the parent nodes and not the leaf nodes?. If so, please guide me how to do so.

Thanks

Gowri

»
Login to post comments

Link

Submitted by Athen OShea on Fri, 06/17/2011 - 13:36.

I believe you'll find the updated Checkbox Tree here:

http://www.essi-lab.eu/projectsSites/lablib-checkboxtree/download.html

Cheers,

Athen O'Shea
Editor, JavaWorld

»
Login to post comments

Need updated download link

Submitted by judy_kennedy on Wed, 06/15/2011 - 15:38.

JW's Most Read

Recent:
10 world-changing technologies for the future
Use == (or !=) to compare Java enums
Google's top priority isn't enterprise IT
Why should users pay for coders' mistakes?
Book review: Java: The Good Parts

From the archives:
Understanding Actor concurrency, Part 1 (2009)
Java concurrency with thread gates (2009)
Hyper-threaded Java (2006)

I tried the download link in the April 8th message, but that didn't work for me either. Is there another link I can try for downloading CheckboxTree?

Thanks.

»
Login to post comments

I had this problem too using

Submitted by Anonymous on Fri, 05/14/2010 - 11:18.

I had this problem too using version 3.2-SNAPSHOT. Using version 3.1.1 fixed this problem.

Otherwise, fantastically useful Swing component guys, thank you!

»
Login to post comments

Error

Submitted by Anonymous on Tue, 04/20/2010 - 15:33.

I tried to use the CheckboxTree but couldn't create one without error. Not even the default constructor worked.

The overridden method
public void setCellRenderer(TreeCellRenderer tcl)..

fails by default as swing buts in the parent (JTree) an invalid default 'TreeCellRenderer' in it...

»
Login to post comments

donwload link

Submitted by Anonymous on Thu, 04/08/2010 - 13:33.

http://ulisse.pin.unifi.it:8081/nexus/index.html#nexus-search;quick~labl...

only select the file that you want download and press the server pom or artifac (i recommended artifac)

sorry my english is bad...

»
Login to post comments

Download

Submitted by Anonymous on Tue, 04/06/2010 - 02:06.

Can anyone provide me the download link please. Not able to find download link on the page.

»
Login to post comments

Download

Submitted by Anonymous on Fri, 02/05/2010 - 06:14.

The download server is down and I really need this checkbox tree, so could somebody give me an alternative link, please?

»
Login to post comments

It rocks !

Submitted by Anonymous on Fri, 01/08/2010 - 10:26.

It rocks !

»
Login to post comments