Introduce the JListView component in JComponentPack 1.1.0

June 16, 2008

If you want to implements the Windows explorer like feature in Java swing application, the **JListView** component meets your requirements exactly.

The **JListView** component support the 5 different view modes: small icon, large icon, list, thumbnails, details, all these view mode can change on the fly, the methods "JListView.setViewMode" can change the view mode of **JListView** component.

The **JListView** component have the MVC design concept, a simple **TableModel** can be provided for it's data, a simple ***CellProvider*** such ***IconProvider*** can be provided for it's icon. Should write a **DefaultCellRenderer** subclass for its renderer and **DefaultCellEditor** subclass for it's editor. The article "Introduce Cell Renderer" introduce why using the **DefaultCellRenderer**.

The **JListView** component use a **ListSelectionModel** as it's selection model, you can change the selection model's mode, it support single selection, single interval selection, multiple interval selection, you can use the following methods to get the selected values:

JListView.getSelectedValue(); // get the lead selection value
JListView.getSelectedValues(); // get all selected values

The **JListView** component provides several methods for it's editing:

JListView.isEditing(); // determines whether the **JListView** is being edited.
JListView.cancelEditing(); // cancel current editing
JListView.stopEditing(); // stop the current editing and apply the editing value
JListView.startEditingAtIndex(); // start the editing at the specified index
JListview.getEditingInex(); // get the current editing object's index

The com.zfqjava.swing.model and **com.zfqjava.swing.cell** package have the **FileTableModel** and FileProvider, it support the directory list and file icon directly, the following code can create a explorer like GUI:

```
JListView listView = new JListView();
listView.setListData(new FileTableModel(new File(System.getProperty("user.home"))));
listView.setCellRenderer(new FileCellRenderer());
listView.setCellEditor(new FileCellEditor());
```

The **JListView** component also support row sorting, the **TableModel** you provided for **JListView** only need implements the ColumnSorter interface, it can support the row sorting automatically, we want to improve this area after upgrade the JRE version to 1.6.

The **JListView** component provides several important client property:

"JListView.rowSelectionAllowed" allow the full row can be selected
"JListView.showVerticalLines" shows the vertical lines in details view mode.
"JListView.showHorizontalLines" shows the horizontal lines in details view mode.
"JListView.backgroundImage" sets the background image for **JListView** component.
For details, you can view the **JListView** JavaDoc API documentation.

The **JListView** also support the Drag and Drop, but in JComponentPack 1.1.0 and early version, implements this feature has trick and tips:

```
// get JTable and JList
BasicListViewUI ui = (BasicListViewUI)listView.getUI();
JTable table = ui.getTable();
JList list = ui.getList();
table.setDragEnabled(true);
list.setDragEnabled(true);
TransferHandler th = new TransferHandler() {
    public int getSourceActions(JComponent c) {
        return COPY;
    }
    protected Transferable createTransferable(JComponent c) {
        // just a test
        Object o = listView.getSelectedValue();
        if(o != null) {
        return new StringSelection(o.toString());
        }
        return null;
    }
    };
table.setTransferHandler(th);
list.setTransferHandler(th);
```

In the upcoming version JComponentPack 1.2.0, we have improved this area, so in the new version, implements the drag and drop feature is very simple:

```
listView.setDragEnabled(true);
TransferHandler th = new TransferHandler() {
    public int getSourceActions(JComponent c) {
        return COPY;
    }

    protected Transferable createTransferable(JComponent c) {
        // just a test
```

```
            Object o = listView.getSelectedValue();
            if(o != null) {
            return new StringSelection(o.toString());
            }
            return null;
        }
        };
    listView. setTransferHandler(th);
```