TreeTableColumn

Lets see how we can modify columns in the tree.

How to use it?

Here is what is really interesting about the JContextTree, making your own columns.
You perhaps allready experienced the hard work related to creation of columns in JTable ... so you think the same way now. But NO! I made things simple for you. lets go see how that works.

```
        //Manipulating columns-----------------------------------------------
        TreeTableColumn mycolumn = buildColumn();

        tree.addColumn(mycolumn);
//        tree.removeColumn(mycolumn);
//        tree.removeColumn(0);
//        int nbcolumns = tree.getColumnCount()
//        int index = tree.getColumnIndex(mycolumn);
//        TreeTableColumn[] cols = tree.getColumns();


public TreeTableColumn buildColumn() {

        class MyCellComponent extends RenderAndEditComponent {

            private MapLayer layer = null;
            private JButton button = new JButton("map");
            private boolean edited = false;

            MyCellComponent() {
                setLayout(new GridLayout(1, 1));
                add(button);

                button.addActionListener(new ActionListener() {

                    public void actionPerformed(ActionEvent e) {
                        if (layer != null) {
                            edited = true;
                            JOptionPane.showMessageDialog(null, "hello layer : " +
layer.getTitle());
                        }
                    }
                });

            }

            // we edit our component depending on what is been passed
            // This component is used for all cell so be sure to
            // put back value to default if needed
            @Override
            public void parse(Object obj) {
                removeAll();
                edited = false;

                if (obj instanceof MapLayer) {
                    layer = (MapLayer) obj;
```

```
                    add(button);
                } else {
                    layer = null;
                }
            }

            //   the returned value when edition stop

            @Override
            public Object getValue() {
                return edited;
            }
        }

        class FlyingColumn extends TreeTableColumn {

            public FlyingColumn() {

                //properties of our column
                setWidth(70);
                setPreferredWidth(70);
                setMaxWidth(70);
                setResizable(false);
                // This is interesting, if you set it to true the cell will switch
                // to edit mode on mouseOver. It makes the column much more dynamic
                // but also need a bit more performance
                setEditableOnMouseOver(true);

                //... and many others available...


                //THE RENDERER AND EDITOR COMPONANTS
                //
                // If you are used to CellRenderer and CellEditor you can use :
                // - setCellEditor( myCellEditor );
                // - setCellRenderer( myCellRenderer );
                //
                // If you are not used to, I prepare a simplified solution used below
                //
                setCellEditor(new DefaultCellEditor(new MyCellComponent()));
                setCellRenderer(new DefaultCellRenderer(new MyCellComponent()));
                //
                // The same component type is used for rendering and editing
                //

                //THE HEADER
                setHeaderValue("FreeMap");
            // If you want an icon in the header use :
            // setHeaderValue( new HeaderInfo( "FreeMap_id", "FreeMap", myIcon ) );
            // - "FreeMap_id" will be visible in the topleft control menu of the
JContextTree
            // - "FreeMap" is the title seen in the header, you can use null
            // - myIcon is your Icon, you can use null
            //
            // If you're not satisfied yet you can rewrite the header renderer
            // setHeaderRenderer( myRenderer );

            }
```

```
        @Override
        public void setValue(Object target, Object value) {
            // here the Editor component returns a Value
            // We made it return a Boolean.
            // We'll just set visible or unvisible the

            boolean edited = (Boolean) value;

            if (edited) {
            //do some stuff
            }

        }

        @Override
        public Object getValue(Object target) {
            // we return the MapLayer, but we can return anything
            // just be sure the Renderer and Editor Component will
            // handle it the same way
            if (target instanceof MapLayer) {
                return target;
            } else {
                return null;
            }
        }

        @Override
        public Class getColumnClass() {
            // the class of the object returned
            return MapLayer.class;
        }

        @Override
        public boolean isCellEditable(Object target) {
            // We choose on what kind of Object (MapContext or MapLayer)
            // the component will be visible.
            // Here we choose only MapLayers
            if (target instanceof MapLayer) {
                return isEditable();
            } else {
                return false;
            }
        }
    }

    return new FlyingColumn();
```

```
    }
```