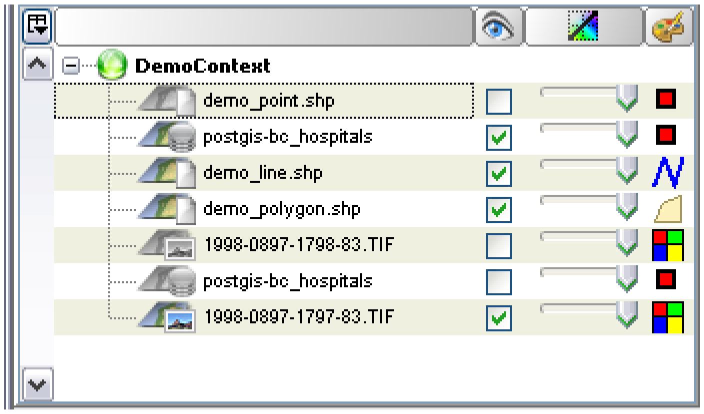
JContextTree



JContextTree

As you can see the JContextTree isn't a JTree nore a JTable, It's a JXTeeTable from library SwingX.

This component is the second most important after the JMapPane.
This tree is an independent widget, it's also a bean, so you can add it in the Swing Palette of NetBeans Matisse WYSIWYG for example.

UseCase/Tutorial

Complete demo class :
http://svn.geotools.org/geotools/trunk/gt/demo/example/src/main/java/org/geotools/demo/widgets/Demo_ContextTree.java

- How to create my own column ?
- How to create my own menu item ?
- How to create my own nodes ?

The JContextTree

Using the usuel constructor :

```
JContextTree tree = new JContextTree();
```

This way you will obtain a one column tree with no popup menu items.

✓ **Do I need to add the JContext to a ScrollPane?**

No need. It's allready in one.

Manage the contexts :

The tree is independant, you have perhaps noticed that the JContextTree class is a frame class, you have no access to the tree model, the column model or the jpopupmenu. You perhaps think that will limit the possibilities to personalize the tree. Yes you're completely right! But that's hard work to keep everything related between the listeners, the drag&Drop suport and the different models. Beside the root component JXTreeTable so I decided (for integrity reason) to lock everything.

Once a MapContext is added in the tree, it will listen to it and update when needed, just manipulate the context like you always do and the tree will fallow.

There are plenty of methods to add/remove/access the contexts in the tree, here are some :

```
tree.addContext(mycontext);
tree.removeContext(mycontext);
tree.getMapContextIndex(mycontext);
...
```

And there is an active context (with a little green icon on the tree) :

```
mycontext = tree.getActiveContext();
tree.setActiveContext(mycontext);
```

It can be used to see on wich context you're working on in your application.

Listen to the tree :

Two different listeners possibles.

listen to the contexts

code is better than explication

```
TreeContextListener treeContextListener =
new TreeContextListener() {

            public void
contextAdded(TreeContextEvent event) {
                System.out.println("Context
added :" + event.getContext().getTitle() + "
at position :" + event.getToIndex());
```

```
            }

            public void
contextRemoved(TreeContextEvent event) {
                System.out.println("Context
removed :" + event.getContext().getTitle() +
" at position :" + event.getFromIndex());
            }

            public void
contextActivated(TreeContextEvent event) {
                System.out.println("Context
activated :" + event.getContext().getTitle()
+ " at position :" + event.getFromIndex());
            }

            public void
contextMoved(TreeContextEvent event) {
                System.out.println("Context
moved :" + event.getContext().getTitle() + "
from : " + event.getFromIndex() + " to : " +
event.getToIndex());
            }
        };


tree.addTreeContextListener(treeContextListe
ner);
//
tree.removeTreeContextListener(treeContextLi
stener);
```

```
//          TreeContextListener[]
treeContextListeners =
tree.getTreeContextListeners();
```

listen to the selection

code is better than explication

```
TreeSelectionListener treeSelectionListener
= new TreeSelectionListener() {

            public void
valueChanged(TreeSelectionEvent e) {

System.out.println("Selection Paths : " +
e.getPaths());

                if (e.getPath() != null) {
                    ContextTreeNode node =
(ContextTreeNode)
e.getPath().getLastPathComponent();
                    Object obj =
node.getUserObject();
                    if (obj instanceof
MapLayer) {

System.out.println("It's a layer node");
                    } else if (obj
instanceof MapContext) {

System.out.println("It's a context node");
                    } else {
```

```
System.out.println("It's something else
node");
                    }
                }
            }
        };


tree.getTreeSelectionModel().addTreeSelectio
nListener(treeSelectionListener);
```

```
//tree.getTreeSelectionModel().removeTreeSel
ectionListener(treeSelectionListener);
```

Copy/Cut/Paste/Delete/Duplicate and the Buffer functions

I take care to add those functions, can be usefull.

When you paste, sometimes the tree will have to "duplicate" the element (when copy/paste in the same context). To separate the origial from the copy he had a prefix to the name.

```
String prefix = tree.getPrefixString();
tree.setPrefixString("It is a copy element -
");
```

You also have differents methods :

```
//see if he can do the action
boolean canCopy = tree.canCopySelection();
boolean canCut = tree.canCutSelection();
boolean canDelete =
tree.canDeleteSelection();
boolean canDuplicate =
tree.canDuplicateSelection();
boolean canPaste = tree.canPasteBuffer();

//do the action, return true if succeed
boolean copySucceed =
tree.copySelectionInBuffer();
boolean cutSucceed =
tree.cutSelectionInBuffer();
boolean deleteSucceed =
tree.deleteSelection();
boolean duplicateSucceed =
tree.duplicateSelection();
boolean pasteSucceed = tree.pasteBuffer();

//and other stuff
boolean selection = tree.hasSelection();
boolean empty = tree.isBufferEmpty();
Object[] datas = tree.getBuffer();
tree.clearBuffer();
```