JAS3 Navigation Tree

Introduction

The built-in FTreePlugin provides the navigation tree that appears on the left side of the JAS3 panel; such tree is meant to be the place where plugins interact with each other and make objects available to the user. Plugins can interact with the navigation tree in several ways:

- add and remove nodes in the tree
- define the appearance and behavior of nodes, like icon, text, popup menus, double click actions etc.
- change the appearance and behavior of existing nodes
- modify the behavior and appearance of nodes added by other plugins

The nodes on the tree have the following main properties:

- a type that is used to define their appearance and behavior. The type is defined once and for all at creation time by the plugin that adds the node to the tree. It cannot be modified.
- a value-key table used by plugins to store objects in the node. When the node is created such table is empty. Any plugin can access this table to either add, retrieve or modify its content.
- space for an object that can be added at creation time by the plugin creating the node. Other plugins can add additional objects to the node by registering object providers with the tree for the given node's type. All these objects can be accessed by type through the node itself.

By default nodes in the tree are pretty boring objects:

- the tree provides a default icon for folders (in their open/close and selected/unselected states) and one for leaf nodes
- the text showing next to the icon is the name of the node, i.e. the last part of their path
- by right clicking on it a minimal popup menu will appear with a few default menu item; currently these default menu items allow the user to sort the node's children (if available) and to rename the node (if possible)

To make nodes more interesting plugins have to register adapters with the tree. Adapters are assigned to nodes based on the node's type; they are used to define the node's appearance and behavior. Through the adapter it is possible to define the following features:

- node's icon for the open/close, selected/unselected node's states
- the text appearing next to the icon; this is an alternative to the node's name
- decide if the node is allowed to have children. If a node is allowed to have children any node will be able to add children to the existing node, otherwise no children will be allowed (when a node is added to a node that does not allow it an exception is thrown, how can it be cought?)
- define the result of a double click action on the node
- define the result of a single click action in the node
- modify the default popup menu
- assign an object to the node that can be accessed by other plugins
- provide a tooltip message to be displayed when the mouse is hovering over the node
- provide a status message to be displayed a the bottom left corner of the JAS3 panel when the node is selected
- define what objects can be transferred in a drag and drop operation
- provide a command processor to define a set of commands to appear on the toolbar; these commands are active when the given node is selected
- decide if the text next to the icon can be edited
- define the selection behavior when one or more nodes are selected
- provide the node's internal structure (see the structure provider below)

When more than one adapter is compatible with a given node's type, they are arranged by the adapter's priority; please refer to the FTreeNodeAdapter's API for a detailed description on how the priority is used for each of the features above.

We will first provide an overview of the main features of the navigation tree; we will then provide a set of examples to show how plugins interact with the navigation tree.

Overview of the main features

We provide here an overview of the main features of the navigation tree. For a detailed list of all its interfaces and classes and their methods please refer to the APIs.

Events and Notifications

Two possible communication channels are available between the tree and the plugins: notifications and events. Notifications are passed to the tree by plugins to inform the tree of what should happen to its structure, while events are sent by the node's tree to the registered listeners when the node's substructure has changed.

* Notifications

A plugin should use notifications when the tree structure is to be changed. Such structural changes involve
o adding and removing nodes
o moving nodes from one folder to another (this operation preserves the original node's information. It is different from removing the node form the original path, adding it back again. This last operation does not preserve the information of the original node as it is lost when the node is removed.)
o renaming a node
o repaint a node or its entire substructure when either icons or text have changed
o changing the current selection by selecting or unselecting a given path

Notifications can be sent to the tree from any thread. Internally they are queued and executed asynchronously on the event dispatching thread. This is to avoid user's programs or scripts having to wait for the GUI to update.

* Events

Events are sent by the nodes on the tree to their registered listeners to communicate one of the following changes to the node's substructure:
o a sub-node has been added or removed
o a sub-node has been changed in the way it appears
o the substructure of a node has changed

By default each node is registered to listen to its children's events. All the events are then propagated back in the tree from children to parent till they reach the tree's root.

The following list shows which events are fired when a notification is processed by the tree
o a node added event is fired every time a node added notification is processed. The parent node is the source of the event and the added node is passed along with the event. If, as a result of a node added notification, more nodes are added to the tree, an event will be fired for each of the added nodes
o a node removed event is fired every time a node removed notification is processed. The parent node is the source of the event with the deleted node being part of the event.
o a node changed event is fired when
+ a node renamed notification is process. In this case the renamed node is the source of the event.
+ a notification that repainted a single node has been processed. The repainted node is the source of the event.
o a node structure changed event is fired when
+ a node's substructure has been repainted by a corresponding notification. The root node of the substructure that has been repainted is the source of the event.
+ a node structure change notification has been processed. Even in this case the root node of the substructure that has changed is the source of the event.
+ as a consequence of a notification that expanded a node. The expanded node is the source of the event.

Cached information

The following information is cached in the nodes after the first time:

- if the node allows children
- the node's name, i.e. the final part of the node's path
- the node's type
- if the node has already checked for its children
- the node's structure provider

Sorting

Nodes can be sorted with an extensible set of sorting algorithms. The following sorting algorithms are provided:

- alphabetical - this is applied to the node's name
- alphanumerically - sorts both the textual part and the numerical part of a node's name; the textual part is sorted alphabetically and the numerical part by value
- folders first - the folders are displayed before the leaf nodes
- default order - nodes are displayed in the order specified by the structure provider (see below). The default ordering is the creation one, i.e. as they were added in the tree.

A plugin can add other sorting algorithms to the set of provided ones; these can be applied alone or in combination with the ones above (for example it is possible to sort the folder first and also apply the alphabetical sorting). The sorting algorithm can be selected by right clicking on any node selecting the "sorting..." menu item; the sorting can be applied to the whole tree or to individual folders, recursively or not.

The nodes are sorted when added to the tree based on the selected combination of sorting algorithms. The default ordering is the one given by the structure provider for the given set of nodes.

Structure Provider

Via the structure provider it is possible to control the structure or a node, i.e. the list of its children and their order. The structure provider for a given node basically holds the list of the node's children. This list is used to display the node's children in the tree in the order in which they are provided. When a node is added the structure provider is asked to add the node to the list of children; the structure provider can decide not to add the children to its list. Similarly when a node is removed from its parent, the parent's structure provider is asked decide whether to remove that node from its list or not.

Adding children on demand (checkForChildren)

A plugin can add nodes to the tree at any time. For example it can add all the nodes required by its needs at once, in the initialization process. If there are several nodes involved in this operation and, especially, if the plugin is accessing the node's information over a network, adding all the nodes during initialization could be a rather time consuming operation. To avoid this waste of time the plugin can alternatively decide to add the nodes on demand, i.e. when they are really needed.

This can be done through the node's adapter. When the tree really needs the information regarding the children of a given node (for example when the node is expanded), it asks all the adapters registered for the give node to check for the node's children. At this point the plugin can add the required nodes, just when they are needed. This information will be required by the tree only once.

The tree provider

The FTreeProvider interface is the access point for the user to the JAS3 navigation tree: it allows to access existing trees, create new ones and access the node adapter registry with which it is possible to register node adapters that define the behavior of nodes on the tree. To tree provider can be obtained through the JAS3 lookup table as shown in the following line:

FTreeProvider treeProvider = (FTreeProvider) getApplication().getLookup().lookup(FTreeProvider.class);

Examples

The following examples show how to interact with the navigation tree; the code provided in each of them can be compiled in run within JAS3.

Adding and removing FTreeNodes

To add a node to the FTree it is necessary to send an FTreeNodeAddedNotification to the tree specifying the type of the node and its location within the tree (optionally it is possible to provide the object contained in the node). The path can either be a String in the standard unix notation (directories separated by "/") or an FTreePath. The type defines the behavior of the node; two default type are provided: FTreeLeafNode for leaves in the tree, and FTreeFolderNode for folders.

tree.treeChanged(new FTreeNodeAddedNotification(this, "/baseDir/subDir", FTreeLeafNode.class));

When adding a node to the tree, intermediate folders will be automatically created if missing. Try the following code for a complete example.

AddNodesToTree.java

```
import org.freehep.application.studio.Studio;
import org.freehep.jas.plugin.tree.FTree;
import org.freehep.jas.plugin.tree.FTreeFolderNode;
import org.freehep.jas.plugin.tree.FTreeLeafNode;
import org.freehep.jas.plugin.tree.FTreeNodeAddedNotification;
import org.freehep.jas.plugin.tree.FTreePath;
import org.freehep.jas.plugin.tree.FTreeProvider;
import org.freehep.util.FreeHEPLookup;

public class AddNodesToTree {


    public static void main(String[] args) {

        // Get the Studio, the FTreeProvider and the default FTree
        Studio studio = (Studio) Studio.getApplication();
        FTreeProvider treeProvider = (FTreeProvider) studio.getLookup().lookup(FTreeProvider.class);
        FTree tree = treeProvider.tree();

        // Add nodes using FTreePath
        FTreePath path = new FTreePath("baseDir");
        tree.treeChanged( new FTreeNodeAddedNotification(studio, path, FTreeFolderNode.class) );
        tree.treeChanged( new FTreeNodeAddedNotification(studio, path.pathByAddingChild("subDir-a"),
FTreeLeafNode.class) );
        path = path.pathByAddingChild("subDir-b").pathByAddingChild("subSubDir-a");
        tree.treeChanged( new FTreeNodeAddedNotification(studio, path, FTreeLeafNode.class) );

        // Add nodes using strings
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "baseDir-1", FTreeFolderNode.class) );
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "baseDir-1/subDir-a", FTreeLeafNode.class) );
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "baseDir-1/subDir-b/subSubDir-a",
FTreeLeafNode.class) );
    }

}
```

To remove an FTreeNode from the tree use an FTreeNodeRemovedNotification as shown in the following code.

RemoveNodesFromTree.java

```
import org.freehep.application.studio.Studio;
import org.freehep.jas.plugin.tree.FTree;
import org.freehep.jas.plugin.tree.FTreeFolderNode;
import org.freehep.jas.plugin.tree.FTreeNodeRemovedNotification;
import org.freehep.jas.plugin.tree.FTreePath;
import org.freehep.jas.plugin.tree.FTreeProvider;
import org.freehep.util.FreeHEPLookup;

public class RemoveNodesFromTree {


    public static void main(String[] args) {

        // Get the Studio, the FTreeProvider and the default FTree
        Studio studio = (Studio) Studio.getApplication();
        FTreeProvider treeProvider = (FTreeProvider) studio.getLookup().lookup(FTreeProvider.class);
        FTree tree = treeProvider.tree();

        // Remove nodes using FTreePath
        FTreePath path = new FTreePath("baseDir");
        path = path.pathByAddingChild("subDir-b").pathByAddingChild("subSubDir-a");
        tree.treeChanged( new FTreeNodeRemovedNotification(studio, path) );

        // Remove nodes using strings
        tree.treeChanged( new FTreeNodeRemovedNotification(studio, "baseDir-1/subDir-b") );
    }

}
```

Adding nodes on two navigation trees

As mentioned earlier the tree provider allows the user to create new navigation tree; the following code shows how to create two navigation trees adding nodes to them.

TwoNavigationTrees.java

```
import org.freehep.application.studio.Studio;
import org.freehep.jas.plugin.tree.FTree;
import org.freehep.jas.plugin.tree.FTreeFolderNode;
import org.freehep.jas.plugin.tree.FTreeLeafNode;
import org.freehep.jas.plugin.tree.FTreeNodeAddedNotification;
import org.freehep.jas.plugin.tree.FTreePath;
import org.freehep.jas.plugin.tree.FTreeProvider;
import org.freehep.util.FreeHEPLookup;

public class TwoNavigationTrees {


    public static void main(String[] args) {

        // Get the Studio, the FTreeProvider and the default FTree
        Studio studio = (Studio) Studio.getApplication();
        FTreeProvider treeProvider = (FTreeProvider) studio.getLookup().lookup(FTreeProvider.class);
        FTree tree1 = treeProvider.tree("tree1");
        FTree tree2 = treeProvider.tree("tree2");

        // Add nodes to the first tree
        tree1.treeChanged( new FTreeNodeAddedNotification(studio, "baseDir-1", FTreeFolderNode.class) );
        tree1.treeChanged( new FTreeNodeAddedNotification(studio, "baseDir-1/subDir-a", FTreeLeafNode.class) );
        tree1.treeChanged( new FTreeNodeAddedNotification(studio, "baseDir-1/subDir-b/subSubDir-a",
FTreeLeafNode.class) );

        // Add nodes to the second tree
        tree2.treeChanged( new FTreeNodeAddedNotification(studio, "/home", FTreeFolderNode.class) );
        tree2.treeChanged( new FTreeNodeAddedNotification(studio, "/home/file.txt", FTreeLeafNode.class) );
        tree2.treeChanged( new FTreeNodeAddedNotification(studio, "/home/tmp/out.out", FTreeLeafNode.class) );
    }

}
```

Create adapter to define double click on tree node

In the above examples the nodes added on the tree had very little functionality. The code in this example shows how to create an adapter to add the double click action to the leaf nodes. In the example we create a new node adapter by extending the DefaultFTreeNodeAdapter, the default implementation of the FTreeNodeAdapter, overwriting its doubleClick method. When double clicking on the nodes created when running the example an empty editor window will appear whose title is the node's name.

BasicDoubleClick.java

```
import java.io.File;
import org.freehep.application.studio.Studio;
import org.freehep.jas.plugin.tree.DefaultFTreeNodeAdapter;
import org.freehep.jas.plugin.tree.FTree;
import org.freehep.jas.plugin.tree.FTreeNode;
import org.freehep.jas.plugin.tree.FTreeNodeAdapter;
import org.freehep.jas.plugin.tree.FTreeNodeAddedNotification;
import org.freehep.jas.plugin.tree.FTreeProvider;
import org.freehep.jas.services.TextEditorService;
import org.freehep.util.FreeHEPLookup;

public class BasicDoubleClick {

    public static void main(String[] args) {

        // Get the Studio, the FTreeProvider and the default FTree
        Studio studio = (Studio) Studio.getApplication();
        FTreeProvider treeProvider = (FTreeProvider) studio.getLookup().lookup(FTreeProvider.class);
        FTree tree = treeProvider.tree();

        // Get the text editor service
        TextEditorService textEditor = (TextEditorService) studio.getLookup().lookup(TextEditorService.class);

        // Register the adapter for nodes of type MyFile.class
        BasicDoubleClick doubleClick = new BasicDoubleClick();
        treeProvider.treeNodeAdapterRegistry().registerNodeAdapter( doubleClick.adapter(textEditor), MyFile.
class );

        // Add nodes to the first tree
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "/home/file.txt", MyFile.class) );
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "/home/tmp/out.out", MyFile.class) );
    }

    FTreeNodeAdapter adapter( TextEditorService service ) {
        return new MyFileAdapter(service);
    }

    private class MyFile {
    }

    private class MyFileAdapter extends DefaultFTreeNodeAdapter {

        private TextEditorService textEditor;

        MyFileAdapter(TextEditorService textEditor) {
            super(100);
            this.textEditor = textEditor;
        }

        public boolean doubleClick(FTreeNode node) {
            String nodeName = node.path().getLastPathComponent();
            textEditor.show(nodeName,"txt", nodeName );
            return true;
        }


    }

}
```

Adding a new sorting algorithm

In the following code a new sorting algorithm is added to the JAS3 lookup table. Running the example a few nodes will be added to the JAS3 navigation tree. By right clicking on the "sortable nodes" folder and selecting the Sorting... menu item you will notice the presence of the algorithm we just added: "Name-length". It sorts nodes based on the length of their name. To achieve this all we had to do was to implement the FTreeNodeSorter interface and register an instance with the lookup table.

AddSortingAlgorithm.java

```
import org.freehep.application.studio.Studio;
import org.freehep.jas.plugin.tree.FTree;
import org.freehep.jas.plugin.tree.FTreeFolderNode;
import org.freehep.jas.plugin.tree.FTreeLeafNode;
import org.freehep.jas.plugin.tree.FTreeNode;
import org.freehep.jas.plugin.tree.FTreeNodeAddedNotification;
import org.freehep.jas.plugin.tree.FTreeNodeSorter;
import org.freehep.jas.plugin.tree.FTreeProvider;

public class AddSortingAlgorithm implements FTreeNodeSorter {


    public static void main(String[] args) {

        // Get the Studio, the FTreeProvider and the default FTree
        Studio studio = (Studio) Studio.getApplication();
        FTreeProvider treeProvider = (FTreeProvider) studio.getLookup().lookup(FTreeProvider.class);
        FTree tree = treeProvider.tree();

        // Add some nodes
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "sortable nodes", FTreeFolderNode.class) );
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "sortable nodes/node 1", FTreeLeafNode.class)
);
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "sortable nodes/abcdefgh", FTreeLeafNode.
class) );
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "sortable nodes/x", FTreeLeafNode.class) );
        tree.treeChanged( new FTreeNodeAddedNotification(studio, "sortable nodes/the longest of them",
FTreeLeafNode.class) );

        // Add the sorting algorithm to the lookup table.
        studio.getLookup().add( new AddSortingAlgorithm() );

    }

    public String algorithmName() {
        return "Name-Length";
    }

    public String description() {
        return "Sorts nodes based on the length of their name.";
    }

    public int sort(FTreeNode node1, FTreeNode node2) {
        return node1.path().getLastPathComponent().length() - node2.path().getLastPathComponent().length();
    }

}
```