NetBeans Platform

Tom Wheeler

11/11/09

SIUE :: November 2009

Intro: What I Cover in this Section

- What a node is (and isn't)
- The various explorer views
- Node activation, lookup and actions
- How NetBeans got its name
- How to create child nodes
- How to create node properties

What is A Node?

- A wrapper for another object
- Adds *presentation* characteristics
  - Icon, actions, display name, description, etc.
  - Properties
  - Child Nodes
- It is NOT related to Java's TreeNode class
  - It's also not a visual component
  - Or a place to store complex data

Nodes and UI Components

- The Explorer API contains UI components
- You can show a node and its children in
  - JList (ListView)
  - JTree (BeanTreeView)
  - JComboBox (ChoiceView)
  - Grid view (IconView)
  - Tree Table (TreeTableView)
  - Menu (MenuView)

Nodes and UI Components

- All the preceding classes
  - Are in the org.openide.explorer package
- Generally easy to switch between them
  - Unlike Swing, you don't have to do much

DEMO: Nodes in the IDE

- I'll show various places where nodes are used in the NetBeans IDE

Nodes and Selection

- Activated nodes
  - Each TopComponent has one or more
  - Might represent the selected file(s)
  - TopComponents can react to changes

Nodes Have A Lookup

- A node has a Lookup
  - You can access via node.getLookup()
- What might you find in a node's Lookup?
  - The node itself
  - A data object representing a file
  - The capabilities (cookies) that node supports
    - Can you save it?
    - Can you print it?

Nodes Have A Lookup (2)

- That last point was important
  - The capabilities (cookies) that node supports
    - Can you save it?
    - Can you print it?
  - Certain things, like actions, react to changes in the lookup of the selected nodes
    - e.g. Save action is only available when something "saveable" is selected
    - How does it know?
    - Let me explain Cookies...

Nodes Have Actions

- Nodes can provide a set of actions
  - These allow user to “do” something
  - Examples might be view, print, save, etc.
- These are rendered into a context menu
- To supply them
  - Override getActions(boolean)
  - Return array of Action
  - Null values become menu separators

Nodes Have Children

- Each node has a Children object
  - A factory for creating child nodes
  - May not actually create any (leaf node)
- Various ways of creating this
  - Children.LEAF (has no child nodes)
  - Children.Keys (created based on collection)
  - ChildFactory + Children.create()
    - Newer, w/ support for background computation

Nodes Have Properties

- Nodes support properties
  - Show information about the node
  - For example, a node representing a file
    - Might have size, last modified date, etc.
  - A node representing a person
    - Might have name, birthday, height, etc.

Nodes Have Properties (2)

- May be grouped into multiple sets
  - Such as “basic” and “advanced”
- May allow reading, writing or both
- Are displayed on the property sheet

Nodes and Beans

- A JavaBean is a design pattern
  - Not related to Enterprise JavaBeans (EJB)
  - Very popular in Java's early days
  - Public no-arg constructor
  - Public fields or discoverable getters/setters
- NetBeans originally used them heavily
  - Node and Node.Property extend java.beans.FeatureDescriptor
  - This is where the name NetBeans came from

Creating Nodes

- Can extend Node class
  - But a lot of work
- BeanNode is a wrapper for JavaBeans
  - Easy if you already have a Bean
- DataNode is a node for DataObjects
  - We have not discussed DataObjects yet
- Better to extend AbstractNode
  - Not actually an abstract class!

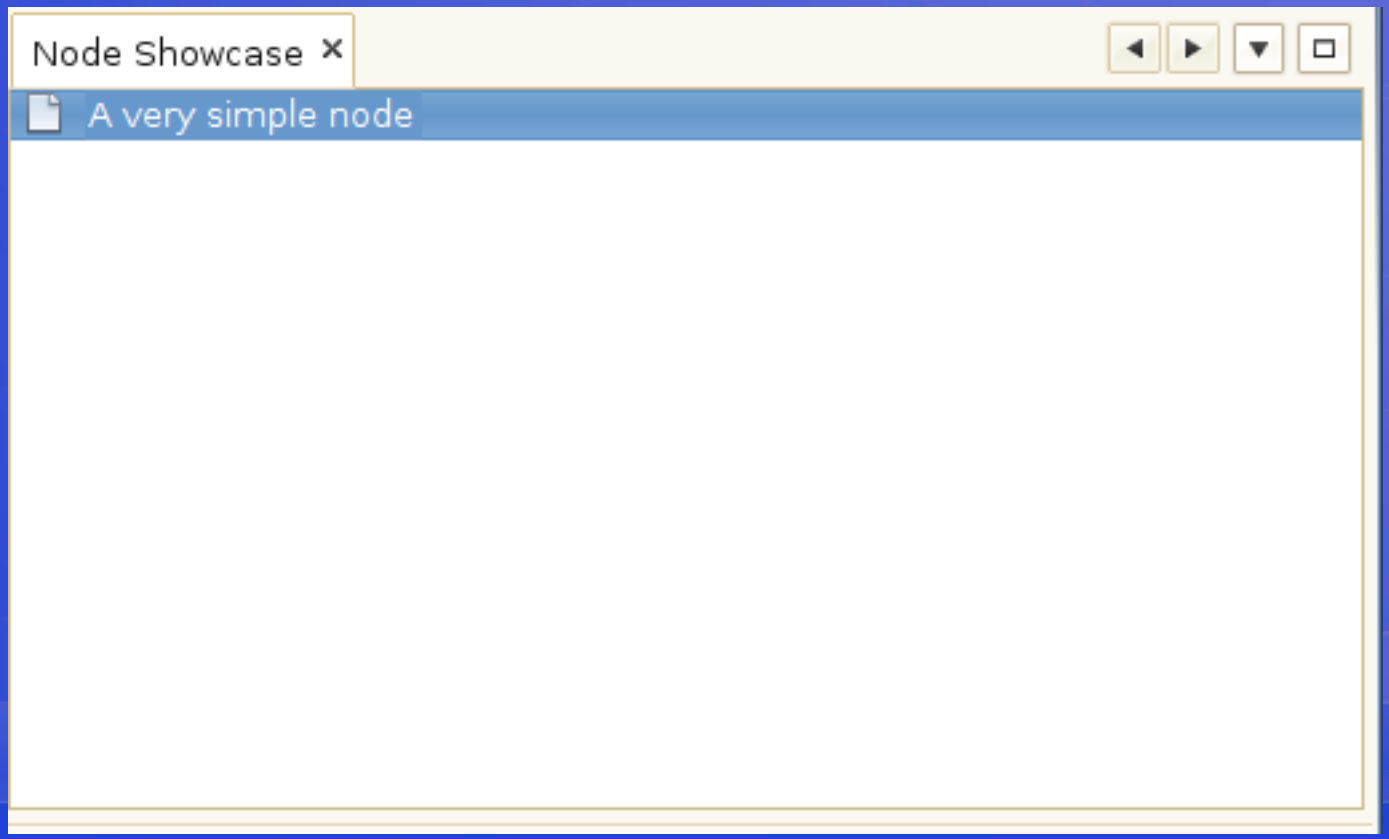
A Simple Node

- Node has no children and no properties

```
package com.tomwheeler.example;

import org.openide.nodes.AbstractNode;
import org.openide.nodes.Children;

/**
 * A very simple Node
 *
 * @author Tom Wheeler
 */
public class SampleNode extends AbstractNode {

    public SampleNode() {
        super(Children.LEAF);
        setDisplayName("A very simple node");
    }
}
```

A Simple Node (View)

- And is rendered like this (BeanTreeView)



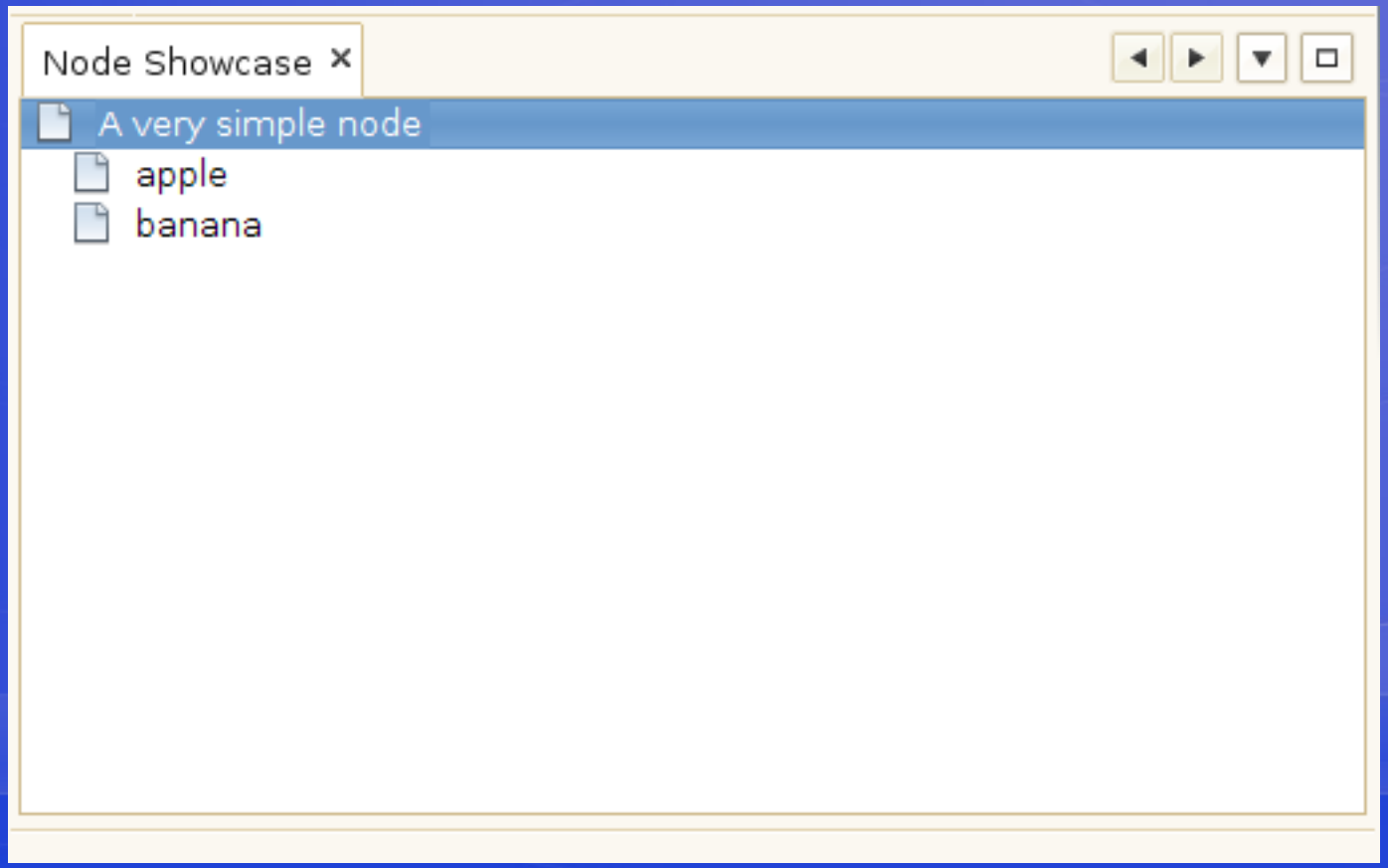
A Node with Children

- This node uses Children.Keys

```
public class SampleNode extends AbstractNode {

    public SampleNode(List<String> childNames) {
        super(new SampleKids(childNames));
        setDisplayName("A very simple node");
    }

    private static class SampleKids extends Children.Keys<String> {

        private List<String> childNames;

        SampleKids(List<String> childNames) {
            this.childNames = childNames;
        }

        @Override
        protected void addNotify() {
            setKeys(childNames);
        }

        @Override
        protected void removeNotify() {
            setKeys(Collections.EMPTY_LIST);
        }

        @Override
        protected Node[] createNodes(String key) {
            Node kid = new AbstractNode(Children.LEAF);
            kid.setDisplayName(key);
            return new Node[]{kid};
        }
    }
}
```

A Node with Children (View)

- And is rendered like this (BeanTreeView)



A Node with Children (A Better Way)

- This node uses ChildFactory

```
public class SampleNode extends AbstractNode {

    public SampleNode(List<String> childNames) {
        super(Children.create(new SampleChildFactory(childNames), true));
        setDisplayName("A very simple node");
    }

    private static class SampleChildFactory extends ChildFactory<String> {

        private List<String> childNames;

        SampleChildFactory(List<String> childNames) {
            this.childNames = childNames;
        }

        @Override
        protected boolean createKeys(List<String> keys) {
            return keys.addAll(childNames);
        }

        @Override
        protected Node createNodeForKey(String key) {
            Node kid = new AbstractNode(Children.LEAF);
            kid.setDisplayName(key);
            return kid;
        }
    }
}
```

A Node with Properties

- Here's a simple node with a property

```
public class SampleNode extends AbstractNode {

    public SampleNode(List<String> childNames) {
        super(Children.LEAF);
        setDisplayName("A very simple node");
    }

    @Override
    protected Sheet createSheet() {
        Sheet sheet = Sheet.createDefault();
        Sheet.Set propertiesSet = Sheet.createPropertiesSet();

        propertiesSet.put(new ExampleProperty("Hello world"));

        sheet.put(propertiesSet);
        return sheet;
    }

    private class ExampleProperty extends PropertySupport.ReadOnly<String> {

        private String value;

        ExampleProperty(String value) {
            super("Greeting", String.class, "Greeting", "What the node says");
            this.value = value;
        }

        @Override
        public String getValue() throws IllegalAccessException, InvocationTargetException {
            return value;
        }
    }
}
```

FilterNode

- FilterNode proxies an existing node
- You can replace aspects of the original
  - Display name
  - Children
  - Actions
  - Icon

Explorer API

- Needed to show the nodes
- Allows other components
  - To determine current selection
  - To react to changes in selection
- Boilerplate code
  - See ExplorerUtils javadoc
  - Hint for a good project
    - Create a wizard to automate this!

Demo: Explorer Views

- Look at explorer view examples
  - http://blogs.sun.com/geertjan/entry/org_openide_explorer_view_the

Review Questions

- Is a node a visual component?
- Name three explorer views
- What is a Cookie?
- What class allows you to change certain aspects of an existing node?
- How is a null value in getActions(boolean) going to be rendered in a context menu?
- What's the purpose of Node.Children?

Recap

- Nodes: an important part of the platform
- They are not visual components
  - Explorer views are displayed based on them
- Nodes have a Lookup
  - And things can listen to it
  - Might contain “capabilities” (cookies)
- Nodes have children
- Nodes have properties

Exercise (45 minutes)

- Create a node with no children
  - Can display using NodeOperation.explore()
    - I will demonstrate this
- Update your node to support children
  - Using Children.Keys or ChildFactory