Jul 25 2014 pyqgis qgis

Recent blog posts

QGIS Layer Tree API (Part 2)

In part 1 we covered how to access the project's layer tree and read its data. Now let's focus on building and manipulating layer trees. We'll also look at how to receive updates about changes within a layer tree.

Starting with an empty project, first we will get access to the layer tree and create some memory layers for testing:

```
root = QgsProject.instance().layerTreeRoot()

layer1 = QgsVectorLayer("Point", "Layer 1", "memory")
layer2 = QgsVectorLayer("Polygon", "Layer 2", "memory")
```

Adding Nodes

Now let's add some layers to the project's layer tree. There are two ways of doing that:

1. Explicit addition. This is done with the `addLayer()` or `insertLayer()` call of the `QgsLayerTreeGroup` class. The former appends to the group node, while the latter allows you to specify the index at which the layer should be added.

```
# step 1: add the layer to the registry, False indicates not to add to the layer tree
QgsMapLayerRegistry.instance().addMapLayer(layer1, False)
# step 2: append layer to the root group node
node_layer1 = root.addLayer(layer1)
```

2. Implicit addition. The project's layer tree is connected to the layer registry and listens for the addition and removal of layers. When a layer is added to the registry, it will be automatically added to the layer tree. It is therefore enough to simply add a layer to the map layer registry (leaving the second argument of addMapLayer() with its default value of `True`):

   ```
   QgsMapLayerRegistry.instance().addMapLayer(layer1)
   ```

   This behaviour is facilitated by the `QgsLayerTreeRegistryBridge` class. By default it inserts layers at the first position of the root node. The insertion point for new layers can be changed – within the QGIS application the insertion point is updated whenever the current selection in the layer tree view changes.

Groups can be added using the `addGroup()` or `insertGroup()` calls of the `QgsLayerTreeGroup` class:

```
node_group1 = root.insertGroup(0, "Group 1")
# add another sub-group to group1
node_subgroup1 = node_group1.addGroup("Sub-group 1")
```

There are also the general `addChildNode()`, `insertChildNode()` and `insertChildNodes()` calls that allow the addition of existing nodes:

```
QgsMapLayerRegistry.instance().addMapLayer(layer2, False)

node_layer2 = QgsLayerTreeLayer(layer2)
root.insertChildNode(0, node_layer2)

node_group2 = QgsLayerTreeGroup("Group 2")
root.addChildNode(node_group2)
```

Nodes that are being added must not have any parent yet (i.e. being part of some layer tree). On the other hand, the nodes that get inserted may already have children, so it is possible to create a whole sub-tree and then add it in one operation to the project's layer tree.

Removing Nodes

The removal of nodes from a layer tree is always done from the parent group node. For example, nodes displayed as top-level items need to be removed from the root node. There are several ways of removing them. The most general form is to use the `removeChildren()` method that takes two arguments: the index of the first child node to be removed and how many child nodes to remove. Removal of a group node will also remove all of its children.

There are several convenience methods for removal:

```
root.removeChildNode(node_group2)

root.removeLayer(layer1)
```

There is one more way to remove layers from the project's layer tree:

```
QgsMapLayerRegistry.instance().addMapLayer(layer1)
```

The project's layer tree is notified when any map layers are being removed from the map layer registry and the layer nodes representing affected layers will be automatically removed from the layer tree. This is handled by the `QgsLayerTreeRegistryBridge` class mentioned earlier.

Moving Nodes

When managing the layer tree, it is often necessary to move some nodes to a different position – within the same parent node or to a different parent node (group). Moving a node is done in three steps: 1. clone the existing node, 2. add the cloned node to the desired place in layer tree, 3. remove the original node. The following code assumes that the existing node we move is a child of the root node:

```
cloned_group1 = node_group1.clone()
root.insertChildNode(0, cloned_group1)
root.removeChildNode(node_group1)
```

Modifying Nodes

There are a number of operations one can do with nodes:

1. **Rename.** Both group and layer nodes can be renamed. For layer nodes this will modify the name directly inside the map layers.

   ```
   node_group1.setName("Group X")
   node_layer2.setLayerName("Layer X")
   ```

2. **Change visibility.** This is actually a check state (checked or unchecked, for group nodes also partially checked) that is associated with the node and normally related to the visibility of layers and groups in the map canvas. In the GUI, the layer tree view is capable of showing a check box for changing the state.

   ```
   print node_group1.isVisible()
   node_group1.setVisible(Qt.Checked)
   node_layer2.setVisible(Qt.Unchecked)
   ```

3. **Change expanded state.** The boolean expanded/collapsed state refers to how the node should be shown in layer tree view in the GUI – whether its children should be shown or hidden.

   ```
   print node_group1.isExpanded()
   node_group1.setExpanded(False)
   ```

4. **Change custom properties.** Each node may have some associated custom properties. The properties are key-value pairs, keys being strings, values being of variant type (QVariant). They can be used by other components of QGIS or plugins to store additional data. Custom properties are preserved when a layer tree is saved and loaded.

   Use the `customProperties()` call to get a list of keys of custom properties, then the `customProperty()` method for getting the value of a particular key. To modify properties, there is a `setCustomProperty()` method which sets a key-value pair and a `removeCustomProperty()` method to remove a pair.

   ```
   node_group1.setCustomProperty("test_key", "test_value")
   ```

```
print node_group1.customProperties()
print node_group1.customProperty("test_key")
node_group1.removeCustomProperty("test_key")
print node_group1.customProperties()
```

Signals from Nodes

There are various signals emitted by nodes which may be used by client code to follow changes to the layer tree. Signals from children are automatically propagated to their parent node, so it is enough to connect to the root node to listen for changes from any level of the tree.

Modification of the Layer Tree Structure

The addition of new nodes always emits a pair of signals – before and after the actual addition. Signals pass information about which node is the parent node and the range of child indices:

- `willAddChildren(node, indexFrom, indexTo)`
- `addedChildren(node, indexFrom, indexTo)`

In order to access the newly added nodes, it is necessary to use the `addedChildren` signal.

The following code sample illustrates how to connect to signals emitted from the layer tree. When the last line is executed, two lines from the newly defined methods should be printed to the console:

```
def onWillAddChildren(node, indexFrom, indexTo):
  print "WILL ADD", node, indexFrom, indexTo
def onAddedChildren(node, indexFrom, indexTo):
  print "ADDED", node, indexFrom, indexTo

root.willAddChildren.connect(onWillAddChildren)
root.addedChildren.connect(onAddedChildren)

g = root.addGroup("My Group")
```

Removal of nodes is handled in a very similar manner to the addition – there is also a pair of signals:

- `willRemoveChildren(node, indexFrom, indexTo)`
- `removedChildren(node, indexFrom, indexTo)`

This time in order to access nodes being removed it is necessary to connect to the `willRemoveChildren` signal.

Modification of Tree Nodes

There are a few more signals that allow monitoring of internal changes to nodes:

- a node is checked or unchecked: `visibilityChanged(node, state)`
- a node's custom property is defined or removed: `customPropertyChanged(node, key)`
- a node gets expanded or collapsed: `expandedChanged(node, expanded)`

Summary

We have covered how to make changes to a layer tree structure and how to listen for changes possibly made by other pieces of code. In a future post we look at GUI components for displaying and modifying the layer tree and the connection between map canvas and layer tree.

Posted by Martin Dobias

info@lutraconsulting.co.uk
|
+44 (0)1444 848012